TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Lehrstuhl für Rechnernetze

Diplomarbeit

ENTWICKLUNG EINES WIEDERVERWENDBAREN FRAMEWORKS ZUR IMPLEMENTIERUNG VON INDOOR/OUTDOOR LOCATION-BASED SERVICES MIT MAPBIQUITOUS

Kay Werner
Mat.-Nr.: 3247807

Betreut durch:
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
und:
Dr.-Ing. Thomas Springer

Eingereicht am 14. März 2012

ZUSAMMENFASSUNG

Aktuelle Betrachtungen im Bereich der Location-Based Services zeigen den großen Nutzen und Zuspruch im Outdoor-Bereich. Soziale Netzwerke stellen diese Dienste ihrer Nutzergemeinde zur Verfügung und auch andere Plattformen integrieren sie immer weiter in ihr Angebot.

In dieser Arbeit wird anfangs ein Überblick über die notwendigen Grundlagen gegeben. Es werden vorhandene Dienste vorgestellt und deren Features untersucht. Diese Analyse macht deutlich, dass die Umsetzung der Dienste für den Indoor-Bereich bisher kaum erfolgte. Es werden Anforderungen an eine solche Umsetzung definiert und begründet. Mit diesen Erkenntnissen wird anschließend ein Framework konzipiert und implementiert, welches bestehende Services mit neuen Indoor-Location-Based Services verbindet. Die Grundlage für das Framework bietet das MapBiquitous-Projekt. Dieses Forschungsprojekt enthält bereits Komponenten zur Indoor-Positionierung und Navigation, die im entstehenden Framework berücksichtigt werden.

Die praktische Umsetzung der Arbeit wird in einer Android-App getestet. Zusätzliche Nutzertests mit der entstandenen App evaluieren abschließend die Funktionalität.

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Systemarchitektur, Lehrstuhl Rechnernetze

AUFGABENSTELLUNG FÜR DIE DIPLOMARBEIT

Name, Vorname:	Werner, Kay
Studiengang:	Medieninformatik
Matr.-Nr.:	3247807
Thema:	**Entwicklung eines wiederverwendbaren Frameworks zur Implemenerierung von Indoor/Outdoor Location-based Services mit MapBiquitous**

ZIELSTELLUNG

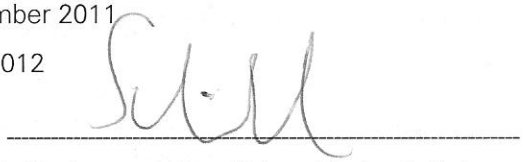
MapBiquitous ist eine Plattform zur Bereitstellung integrierter ortsbezogener Dienste für den Innen- und Außenbereich. Das Konzept beruht auf einer verteilten Infrastruktur von Gebäudeservern, die geometrische Daten der Gebäudestruktur, semantische Gebäudeinformationen, Daten zur Positionsbestimmung im Gebäude und Routinginformationen bereitstellen. MapBiquitous Clients laden die benötigten Gebäudedaten dynamisch und verarbeiten diese clientseitig, um eine Visualisierung von sowie eine Navigation in und zwischen Gebäuden zu ermöglichen. Aufsetzend auf dieser grundlegende Funktionalität von MapBiquitous soll es ähnlich wie bei Google Maps, Bing Maps, etc. möglich sein, Location-based Services zu entwickeln. Beispiele sind etwa die Suche nach dem nächsten Drucker und die Erstellung vor Geotags. Dazu wird ein Konzept zur effizienten Kapselung der Basisfunktionalität von MapBiquitous und zur Bereitstellung dieser Funktionalität als wiederverwendbares Framework benötigt, das es Entwicklern in einfacher Weise ermöglicht, MapBiquitous in ihre Anwendungen einzubinden. Dies betrifft die Aspekte des Zugriffs auf Gebäudedaten, die Integration mit Outdoor-Diensten, die Funktionalität zur Positionsbestimmung, die Visualisierung von Indoor- und Outdoor-Daten sowie die Position von Nutzern und weiteren Ressourcen, und die Erzeugung und Speicherung anwendungsspezifischer Daten (Geotags, KML).

Ziel der Arbeit ist die Erarbeitung eines entsprechenden Konzeptes für die MapBiquitous-Plattform. Dazu sollen bestehende Lösungen wie Google Maps, Bing Maps, etc. hinsichtlich der bereitgestellten Funktionalität und ihrer Struktur untersucht werden, um einerseits die Anforderungen für das MapBiquitous-Framework und andererseits wiederverwendbare Lösungskonzepte zu identifizieren. Darauf aufbauend soll ein plattformunabhängiges Konzept für ein wiederverwendbares MapBiquitous-Framework entwickelt und prototypisch umgesetzt werden. Die erarbeiteten Konzepte sind zu evaluieren und zu bewerten.

SCHWERPUNKTE

- Analyse existierender Frameworks für Location-based Services
- Identifikation der Anforderungen an das MapBiquitous-Framework
- Konzeption eines plattformunabhängigen, wiederverwendbaren Frameworks
- Prototypische Implementierung
- Evaluation und Bewertung der Lösung

Betreuer:	Dr.-Ing. Thomas Springer
Verantwortlicher Hochschullehrer:	Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
Institut:	Institut für Systemarchitektur
Lehrstuhl:	Rechnernetze
Beginn am:	15. September 2011
Einzureichen am:	14. März 2012



Unterschrift des verantwortlichen Hochschullehrers

Verteiler: 1 x HSL, 1 x Betreuer, 1 x Student

INHALTSVERZEICHNIS

1 Einführung **13**

1.1 Motivation 13

1.2 Projekt MapBiquitous 13

1.3 Nutzungsbeispiele 14

2 Grundlagen **15**

2.1 Begrifflichkeiten 15

2.2 Location-Based Services 16

2.2.1 Anwendungsbereiche 18

2.2.2 Klassifizierung 20

2.3 OpenGIS 21

2.3.1 Geodaten 22

2.3.2 Geography Markup Language 23

2.3.3 Features 23

2.3.4 Web Services 23

2.3.5 Filter Encoding 27

2.4 MapBiquitous 28

2.5 Zusammenfassung 31

3 Verwandte Arbeiten 33

3.1 Location-Based Services am Beispiel eines Lokalführers 33

3.1.1 Lokaldetails 33

3.1.2 Wichtige Funktionen 35

3.1.3 Fazit 35

3.2 Google Maps 36

3.2.1 Android App 37

3.2.2 Desktopversion 38

3.2.3 Developer API 38

3.3 Bing Maps 43

3.3.1 Android App 43

3.3.2 Developer API 44

3.4 Yahoo! Maps 45

3.4.1 Android App 45

3.4.2 Developer API 46

3.5 ape@map 46

3.5.1 Konzeption 47

3.6 OpenStreetMap 48

3.6.1 Developer API 48

3.7 Communities 48

3.8 Kritiken 50

4 Anforderungen 51

4.1 technische Umsetzung, Architektur 51

4.1.1 Zusammenfassung 52

4.2 Schnittstelle zum Server, Dienstdefinition 53
4.2.1 Zusammenfassung 54
4.3 UI des Clients, Darstellung 54
4.3.1 Zusammenfassung 55
4.4 Zielgruppe 55
4.4.1 Zusammenfassung 55
4.5 Zusammenfassung 55

5 Konzeption des Frameworks 57

5.1 Datenstruktur 58
5.1.1 Erweiterung der Datenbank 58
5.1.2 Client-seitige Erweiterungen 59
5.2 Schnittstellen / Architektur 59
5.2.1 Client-Architektur 60
5.2.2 Konfiguration des Servers 63
5.2.3 Kommunikation mit dem Server 64
5.3 Darstellung 66
5.3.1 Interaktionsmöglichkeiten 66
5.3.2 Mockups 69
5.3.3 Farbgestaltung 73
5.4 Zusammenfassung 75

6 Implementierung 77

6.1 Datenstruktur 77
6.1.1 Datenbank 77
6.1.2 Serverkonfiguration 79
6.1.3 Modellstruktur 80

6.2 Schnittstellen 85
6.2.1 Beispiel „Find nearest!“ 85
6.3 Darstellung 87
6.3.1 Interaktionsmöglichkeiten 87
6.3.2 Metainformationen und Zusatzfunktionen 88
6.3.3 Datenmanagement 90
6.3.4 Suche und Ergebnisanzeige 91
6.3.5 Navigation 91
6.3.6 Layout 92
6.4 Probleme 92
6.4.1 Client-seitig 92
6.4.2 Serverseitig 93
6.5 Zusammenfassung 93

7 Evaluierung **97**

7.1 UI-Test 98
7.1.1 Testaufbau 98
7.1.2 Testmethoden 98
7.1.3 Auswertung 101
7.1.4 Auswertung der System Usability Scale 101
7.1.5 Bewertung der einzelnen Bestandteile 102
7.1.6 Kritikpunkte und Verbesserungsvorschläge 103
7.2 Framework 103
7.3 Fazit 104

8 Zusammenfassung und Ausblick **107**

8.1 Zusammenfassung 107

8.2 Ausblick 108

A Evaluation **111**

A.1 Fragebogen AB Test 111

A.2 Vergleichsfolien des AB Tests 113

A.3 Aufgaben und Fragen der Abschlussevaluation 116

A.4 Protokoll der Abschlussevaluation 119

A.5 javadoc.xml 121

1 EINFÜHRUNG

1.1 MOTIVATION

Location-Based Services begegnen uns in unserem Alltag an vielen Stellen. Ob es die Suche nach einem geöffneten Geschäft oder die Navigation zur nächsten Sehenswürdigkeit ist, wir nutzen die ortsbezogenen Dienste meist ohne sie direkt wahrzunehmen. Viele Services werden noch über das Internet vom Desktop-PC abgefragt. Die Verbreitung des mobilen Internets und damit der Ausbau der Location-Based Services für unterwegs schreitet aber weiter voran. Der aktuelle Nutzen ist mit der SMS-Abfrage des Busfahrplans schon fast veraltet, da die Smartphoneverbreitung rasant wächst. Auch zahlreiche entwickelte Apps vereinfachen die Nutzung verschiedenster Dienste immer mehr.

Trotz der vielen Möglichkeiten, die sich den Nutzern erschließen, sind die Dienste bisher nur im Outdoor-Bereich angesiedelt. Es gibt kaum Ansätze, welche die gut entwickelten Funktionalitäten auf den Innenraum von Gebäuden übertragen haben. An dieser Stelle lohnt es sich anzusetzen und Forschungsarbeit zu betreiben.

1.2 PROJEKT MAPBIQUITOUS

Das MapBiquitous-Projekt der Technischen Universität Dresden ist dabei die Grundlage zur Entwicklung eines umfangreichen Systems zur Implementierung von Location-Based Services im Indoor-Bereich. Wichtig ist dabei, dass besonders auf die Integration zwischen Indoor- und Outdoor-Services geachtet wird. Ein fließender, für den Nutzer kaum wahrzunehmender Übergang sollte dabei im Fokus stehen. Das Projekt wird nicht nur für den Nutzer die Vorteile der integrierten Dienste bereitstellen, sondern auch für Entwickler die Arbeit erleichtern. Zukünftige Entwicklungen und Forschungen werden auch weiterhin das MapBiquitous-Projekt nutzen und erweitern. Es muss daher zukunftssicher konzipiert und prototypisch an die Gegebenheiten angepasst werden. Weitere Informationen zum aktuellen Projektstand werden im Kapitel 2.4 aufgezeigt.

1.3 NUTZUNGSBEISPIELE

Um die angestrebten Ziele besser verstehen zu können, werden hier zwei kurze Anwendungsbeispiele präsentiert. Eines aus Sicht des Nutzers, der die entstandenen Services zu seinen Gunsten verwenden will. Eines aus Sicht des Entwicklers, welcher das entstehende Framework als Entwicklungsgrundlage nutzen will.

Der Nutzer ist in diesem Beispiel ein neuer Student in einer für ihn unbekannten Stadt. Er hat seinen ersten Tag an der fremden Universität gerade begonnen und benötigt nun dringend Hilfe, um den Vorlesungssaal zu finden. Die Zeit drängt, andere Erstsemester können ihm nicht helfen. Glücklicherweise hat er sein Smartphone mit der MapBiquitous-App dabei. Die Suche nach der Vorlesung zeigt ihm als Ergebnis sofort den Raum und das Gebäude auf dem Campus der Uni. Seinen ebenso orientierungslosen Freunden sendet er noch schnell die Koordinaten, bevor er sich auf dem kürzesten Weg zur Vorlesung navigieren lässt. Nach der Vorlesung sind noch einige organisatorische Dinge zu klären, kurz: er braucht einen Kopierer, den er innerhalb der 20-Minuten-Pause erreichen kann. Wieder kann ihn die schnelle Umgebungssuche retten. Nach der Uni entdeckt der Student mit seinen Freunden bei der Campuserkundung eine gemütliche Sitzecke und wertet mit ihnen den Tag aus. Das sonnige Plätzchen wird gleich in MapBiquitous als Ort zum Entspannen für alle anderen Nutzer geteilt und dient seither als nachmittäglicher Treffpunkt der Gruppe.

Der Entwickler verfolgt eine Idee: Er will eine Software entwickeln, welche die Verwaltung flexibler Arbeitsplätze in einen Großraumbüro übernimmt. Dazu sollen sich die Arbeiter mit ihrem Smartphone am Arbeitsplatz einloggen und diesen damit als „belegt“ markieren. Andere neu ankommende Arbeiter suchen sich mit der App den nächsten freien Platz und lassen sich direkt an den Ort leiten. So sollen sich auch Kollegen, die jeden Tag einen anderen Platz besetzen, finden lassen und schnelle Treffpunkte festgelegt werden können. Für Ortung, Navigation und Suche braucht er daher ein stabiles Framework, mit welchem er auch seine zusätzlichen Wünsche implementieren kann. Er stößt bei Recherchen auf MapBiquitous und fängt sofort an zu programmieren.

2 GRUNDLAGEN

2.1 BEGRIFFLICHKEITEN

In dieser Arbeit werden viele Fachbegriffe verwendet, die sich dem Leser nicht auf Anhieb erschließen. Dieser Abschnitt dient der Erläuterung der Begriffe, welche im gesamten Text Verwendung finden und für das Verständnis relevant sind. Um die Bedeutung aller Ausdrücke einheitlich zu klären und dem Leser von Anfang an ein Grundwissen zu vermitteln, folgt eine Auflistung grundlegender Termini. Sie sind nach ihren Zusammenhängen miteinander geordnet:

LBS ist die Abkürzung für *Location-Based Service*, siehe 2.2

Position wird in der Arbeit gleichbedeutend mit den Koordinaten eines Ortes verwendet. Wenn von Positionsbestimmung oder ähnlichen in Zusammenhang stehenden Begriffen die Rede ist, wird davon ausgegangen, dass die Koordinaten des beschriebenen Ortes bestimmt werden.

Location (Lage) wird als Äquivalent zu Position oder Ort verwendet. Sofern ein „allgemeiner" Ort, wie etwa ein Marktplatz oder ein Restaurant gemeint ist, wird dies im Text verdeutlicht.

Fingerprinting ist eine Methode zur WLAN-Positionierung. Mit Hilfe von vorher kartierten WLAN-Signalstärken, kann zur Laufzeit ein Vergleich mit aktuellen Messwerten erfolgen. Die beste Übereinstimmung mit den vorhandenen Daten lässt dann Rückschlüsse auf die Position zu.

Crowdsourcing bezeichnet das Generieren von Daten durch viele Nutzer. Die Nutzer sind meist Freiwillige, die ihre Arbeitszeit zur Verfügung stellen.

Point of Interest ist ein „interessanter Ort". Diese Orte bezeichnen besonders markante Punkte zu denen weitere Informationen in einer Karte verfügbar sind.

API (engl. application programming interface) ist eine Programmierschnittstelle. Sie ermöglicht es dem Entwickler Funktionen und Dienste anderer Programme zu nutzen.

Service ist ein Dienst der von einem Server zur Verfügung gestellt wird.

Client bezeichnet in dieser Arbeit den Nutzer eines Services.

Request/Response ist die Anfrage an einen Server bzw. die Antwort des Servers an den Client.

UI (engl. user interface) oder auch Benutzeroberfläche. Sie stellt den sichtbaren Programmteil für den Nutzer dar. Der Nutzer interagiert mit der Benutzeroberfläche.

UX (engl. user experience) ist die Erfahrung, die der Nutzer mit einem Programm macht. Sie ist von der UI abzugrenzen. Ein Programm mit ausgereifter UI sollte aber auch eine gute UX bieten, d.h. der Nutzer sollte sich bei der Nutzung wohlfühlen.

App ist die Kurzform von Applikation (engl. Application) oder Anwendung. Damit ist im Rahmen der Arbeit eine bedienbare Software oder ein Programm gemeint, welche auf einem Computer läuft.

Smartphone wird gleichbedeutend für die Bezeichnung *mobiles Endgerät* genutzt. Das Smartphone ist dabei eines von vielen möglichen mobilen Geräten. Da die Unterscheidung in den vorliegenden Fällen nicht von Bedeutung ist, werden die Begriffe gleichberechtigt verwendet. Mit diesem Begriff wird durchgängig ein Gerät bezeichnet, welches für Endnutzer frei verfügbar ist und daher keine spezielle Forschungs- oder Messhardware einschließt.

2.2 LOCATION-BASED SERVICES

Location-Based Services, oder kurz LBS, existieren schon seit längerer Zeit. Sie haben in den letzten Jahren durch die Entwicklung neuerer Techniken aber sprunghaft an Bedeutung gewonnen. Allgemein übersetzt man LBS im Deutschen als ortsbasierte Dienste. Die Begriffe *Location-Based Services* und *ortsbasierte Dienste* werden daher als Synonyme verwendet.

Nach [JW08] bezeichnen LBS die Gesamtheit aller Dienste, welche die geografische Lage eines oder mehrerer Objekte nutzen und verarbeiten. Dabei können die Objekte menschlicher oder nicht-menschlicher Natur sein. Es kann sich also beispielhaft um die Position von Personen oder die Lage von Gegenständen jeglicher Art handeln. In einem ortsbasierten Dienst sind immer zwei Orte involviert. Beide haben einen relativen Abstand voneinander, der ohne Weiteres berechnet werden kann. Das bedeutet, dass sowohl das anfragende, den Dienst nutzende Objekt als auch das gesuchte Objekt durch den Service in Verbindung gebracht werden. Die Objekte können sich unabhängig voneinander in Bewegung befinden oder statisch an einem Ort verharren. Sobald sich mindestens ein Objekt bewegt, werden ständige Aktualisierungen der Position für den Dienst wichtig. Je höher die Genauigkeit der Positionsbestimmung ist, um so höher wird die Qualität des angebotenen Services ([DVLK11]).

Ein weiterer Faktor, der die Popularität der LBS unterstützt, ist die Verbreitung von „Low-cost" GPS-Geräten und die dadurch verfügbare computergesteuerte Navigation im Endnutzerbereich (nach [DHdLD10]). Diese technischen Weiterentwicklungen machten ortsbasierte Dienste für die breite Masse zugänglich. Letztendlich wird der Markt für Location-Based Services sogar als „Killer-App" (siehe [JW08]) im mobilen Umfeld bezeichnet. Als Begründung für diese besondere Stellung in der aktuellen Softwareentwicklung wird das exponentielle Wachstum von 2006 bis

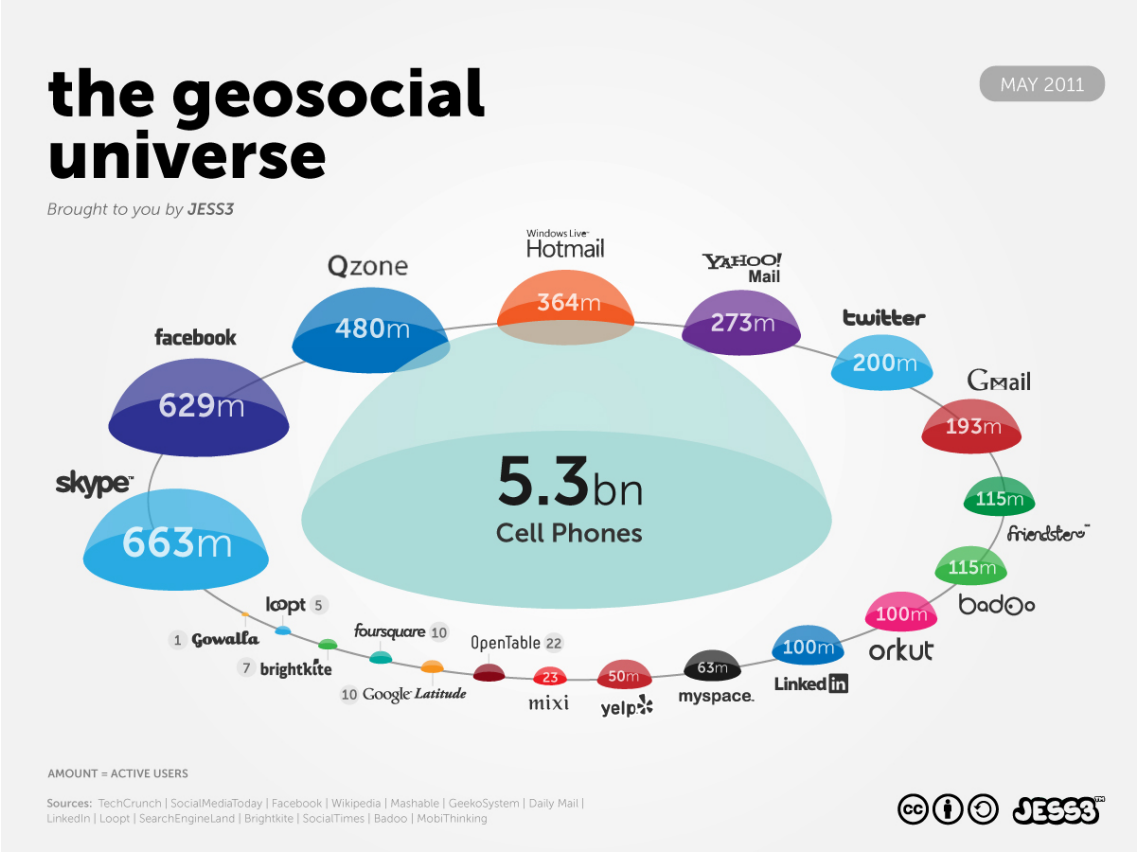


Abbildung 2.1: Verbreitung von Mobiltelefonen und mobilen Anwendungen ([JES11])

2010 herangezogen. Am Beispiel Europa ist in diesem Zeitraum eine Umsatzsteigerung von 191 Millionen Dollar auf 622 Millionen Dollar zu verzeichnen. Die weltweite Verbreitung von Mobiltelefonen und im Vergleich die Nutzung sozialer Netze wird aus der Abbildung 2.1 deutlich. Dabei nutzen die gezeigten Plattformen in unterschiedlichem Maße bereits ortsbezogene Dienste. Netzwerke wie Facebook bieten LBS als Teilfunktionalität an, wobei yelp ausschließlich auf LBS aufbaut. Auf die detaillierten Funktionen der Plattformen wird im Kapitel 3 eingegangen. Dass die Entwicklung in diesem Maße voranschreitet, war 1966 noch nicht abzusehen als die ersten Voraussetzungen für ortsbasierte Dienste in den USA geschaffen wurden. Durch den als „Enhanced 911“ bekannten Beschluss wurden die Forderungen gesetzt, dass eingehende Notrufe mit einer möglichst hohen Genauigkeit geortet werden sollen (siehe [PW11] und [DVLK11]). Die infrastrukturellen Anpassungen erfolgten dann ab Anfang der 80er Jahre. Seit den 90er Jahren gibt es in weiten Teilen der Erde ähnliche Beschlüsse.

Dass das Thema der LBS noch immer ein aktuelles Forschungsthema ist, liegt an den immer neuen Möglichkeiten, die sich erschließen lassen. So werden für private Nutzer das „real-time-mapping“, „searchable directions“ und „POI-contents“ (nach [LMK08], [PW11] und [DVLK11]) immer interessanter. Das bedeutet, dass der Nutzer sich unter anderem mit mobilen Endgeräten aktuelle Kartenpositionen anzeigen lassen und umgebende Örtlichkeiten nach seinen Interessen durchsuchen kann. Weiterhin werden immer noch leistungsfähige Middleware-Frameworks entwickelt, um die Software effektiver und entwicklerfreundlicher zu gestalten ([LMK08]). Das Gebiet der Indoor-Location-Based Services, mit dem sich diese Arbeit umfassend beschäftigt, ist

dabei eines der wichtigsten aktuellen Forschungsthemen, da sich bisher nur wenige Abhandlungen finden lassen. „Indoor" bedeutet in diesem Zusammenhang, dass die Funktionalitäten und Vorteile des jetzigen Entwicklungsstandes auch auf Gebäude übertragen werden sollen. Andere über lange Zeit aktuelle Themen sind die der Datensicherheit und Datenkontrolle (siehe [AM11]). Sie betreffen alle Arten von LBS-Anwendungen, da für die Dienste immer persönliche Daten in hohem Maße versendet und ausgewertet werden. Es gilt hier die richtige Abwägung zwischen freigegebenen Daten und Vertrauen bzw. Risiko des Dienstes zu finden und die Risiken soweit möglich zu minimieren.

Um eine umfassende Zusammenarbeit der einzelnen Dienstanbieter zu ermöglichen, sind eindeutige Standards notwendig. Durch sie werden Schnittstellen zur Verfügung gestellt und Protokolle definiert. Das Vorhandensein einer einheitlichen Standardisierung macht es ebenfalls möglich, neue Netzwerktechnologien und -geräte zu entwickeln und mit ihnen kompatible Software zu implementieren (vgl. [Küp05]). Einige der wichtigsten Unternehmen und Organisationen, welche an den Standardisierungsverfahren beteiligt sind, sind:

3GPP und 3GPP2 (3rd Generation Partnership Project), welche die Entwicklung von technischen Empfehlungen für GSM und UMTS auf internationalem Gebiet unterstützen.

OMA und Parlay (Open Mobile Alliance und Parlay group), die sich auf dem Gebiet der Entwicklung von technischen Spezifikationen zum Datenaustausch engagieren. Sie haben beispielsweise Protokolle für den Austausch von Positionsdaten zwischen zwei Partnern entwickelt.

IETF und W3C (Internet Engineering Task Force und World Wide Web Consortium), die sich um die Integration von LBS und benötigter Daten in Web-basierten Anwendungen bemühen.

OGC (Open Geospatial Consortium), welches Standards für den Umgang mit Geodaten entwickelt.

Da die Vielfalt der Standards zu groß ist um auf alle einzugehen, werden im Kapitel 2.3 die für diese Arbeit wichtigsten genauer beschrieben.

2.2.1 Anwendungsbereiche

Die Anwendungsbereiche der LBS sind sehr umfassend und vielseitig. So lassen sich die unterschiedlichen Services im Unternehmensbereich, direkt für den privaten Endanwender aber auch im öffentlichen Sektor sinnvoll einsetzen. Alle drei Bereiche sind mit jeweiligen Beispielen in der Tabelle 2.1 aufgezeigt. Es wurden dabei Beispiele gefunden, die speziell auf einen Bereich zugeschnitten sind, aber auch übergreifend und in unterschiedlichen Bereichen einsetzbar sind (nach [Bla07], [GNSW06] und [GFDT05]). Um Zusammenhänge zu verdeutlichen, werden nachfolgend aus den spezialisierten und aus den verallgemeinert einsetzbaren Lösungen Beispiele beschrieben.

Dienst / Service	**Unternehmens-anwendung** (Business Service)	**Endnutzeran-wendung** (Consumer Service)	**öffentlicher Dienst** (Government Service)
Flottenmanagement	x		
Waren- und Gütertracking	x		
Suchdienst		x	
medizinische Überwachung		x	
Spiele		x	
ortsspezifische News		x	
Personentracking		x	
Katastrophenmanagement			x
Notruf- und Rettungswesen			x
Warndienste			x
Parkraumverwaltung			x
Personaleinsatzplanung	x		x
Dokumentation	x		x
Workflowmanagement	x		x
ortsspezifische Information		x	x
Tourismus		x	x
Gelbe Seiten	x	x	
Diebstahlverwaltung	x	x	
Navigation	x	x	x

Tabelle 2.1: Klassifizierung der betrachteten Dienste nach [Bla07], [GNSW06], [GFDT05]

Unternehmensanwendungen (engl. Business Services) sind Softwarelösungen, welche die Organisation der Unternehmensprozesse durch den Einsatz ortsbasierter Dienste erleichtern. Die Dienste sind meist auf das spezielle Unternehmen zugeschnitten. So können zum Beispiel beim *Warentracking* effektiv Lagerbestände überwacht werden. Die Lagerung wird damit günstiger und die Verwaltung einfacher.

Endnutzeranwendungen (engl. Consumer Services) beschreiben Dienste, die den privaten Nutzer in seinem Alltag unterstützen. Darunter zählt ebenfalls Unterhaltungssoftware, die Kernfunktionalitäten von ortsbasierten Diensten nutzt. Die häufigste Verwendung tritt in diesem Bereich bei den ortsbasierten *Suchdiensten* auf. Sie werden von vielen Nutzern zum Finden verschiedener Informationen oder Örtlichkeiten in der unmittelbaren Umgebung genutzt. Ein typisches Beispiel für die Nutzung von Unterhaltungssoftware sind ortsbasierte *Spiele*. Es kann sich dabei etwa um virtuelle Schnitzeljagden wie Geocaching handeln. Die LBS-Nutzung im Endnutzerbereich hat mit dem zunehmenden Einsatz von Smartphones stark zugenommen.

Öffentliche Dienste (engl. Government Services) sind die Dienste, welche von der Regierung oder öffentlichen Einrichtungen zur Verfügung gestellt werden. Es handelt sich hierbei häufig um spezielle Warnsysteme für die Bevölkerung. Das älteste System stellt der *Notruf* dar. Hier ist schon seit Mitte des 20. Jahrhunderts eine möglichst hohe Genauigkeit angestrebt, um den Anrufenden (und seinen Notfall) orten zu können. In gefährdeten Gebieten werden *Warndienste* für Erdbeben, die die Bevölkerung frühzeitig informieren immer populärer.

Bereichsübergreifende Dienste betreffen wie bereits erläutert mindestens zwei der drei Anwendungsbereiche. So kann eine *Personaleinsatzplanung* einem Unternehmen aber auch einer öffentlichen Einrichtung oder einem Rettungsteam erhebliche organisatorische Vorteile verschaffen. Die *Diebstahlverwaltung*, also das Orten und Beobachten wertvoller Gegenstände, kann im privaten Sektor aber auch im Unternehmensbereich sinnvoll eingesetzt werden. Zuletzt ist die *Navigation* ein Service, der in allen Bereichen häufig genutzt wird, um Wege zu verkürzen und Transporte zu optimieren.

2.2.2 Klassifizierung

Nachdem im Kapitel 2.2.1 LBS nach Einsatzgebieten unterschieden wurden, kann man die Services auch nach ihrer grundsätzlichen Funktion unterteilen. Dabei kommt es darauf an, zu welchem Zeitpunkt und unter welchen Umständen der Dienst abgerufen wird. Ausgehend von dieser Unterscheidung entstehen Möglichkeiten des Einsatzes, welche bei der Konzeptionierung und Umsetzung der LBS beachtet werden müssen.

Implizite LBS nach [GNSW06] zeichnen sich durch Echtzeitortung während der Dienstnutzung aus. Das bedeutet, dass der Dienst Funktionen zur Verfügung stellt, die sich auf den aktuellen Standort des aufrufenden Gerätes beziehen. Hierzu zählen alle Angebote, die in der näheren Umgebung dem Nutzer zur Verfügung stehen. Der Nutzer hat die Möglichkeit, diese Menge an Diensten zu durchsuchen. Die örtliche Nähe des Dienstes ist dabei im Allgemeinen auf wenige Kilometer beschränkt, Vorgaben werden in dieser Klassifizierung aber keine gemacht.

Explizite LBS sind das logische Gegenstück zu den *Impliziten LBS*. Sie dienen dem Nutzer um Services an einem bestimmten Ort zu erhalten, an dem er sich zum Zeitpunkt des Aufrufes nicht befindet. Zum Beispiel ist das Antippen einer entfernten Kartenposition ein solcher Service, wenn auf Grund dieser gewählten Position die umliegenden Services dem Nutzer präsentiert werden. Es handelt sich ebenfalls um einen Expliziten LBS, wenn eine Ortung aus technischer Sicht nicht möglich ist und die Position manuell gewählt wird. In den gelieferten Ergebnissen unterscheiden sich die beiden Klassifizierungsmethoden nicht von einander.

Eine weitere Klassifizierung kann man erarbeiten, wenn man die Dienste nach der Art des Aufrufes einteilt (siehe [GFDT05]). Hier werden in ähnlicher Weise stationäre und mobile Services gegeneinander abgegrenzt.

Stationäre LBS dienen dem Erkunden der aktuellen Position und ähneln damit einer Mischung aus Impliziten und Expliziten LBS. Zum besseren Verständnis der Dienste kann man mit den folgenden Fragen bzw. Aussagen entscheiden, ob es sich um einen Stationären LBS handelt:

- What is around?
- Find the nearest...
- Find me...

Die Aussagen dienen dem Erkunden der aktuellen Umgebung. Das ist generell und ohne Einschränkung oder als direkte Suche nach dem nächsten gewünschten Punkt möglich. Eine weitere grundlegende Funktion ist das Suchen der eigenen Position.

Mobile Services hingegen kommen zur Anwendung, wenn sich der Nutzer zum Zeitpunkt der Abfrage bewegt. In dieser Klassifizierung spielt der Navigationscharakter eine große Rolle. Es gibt auch hierzu entsprechende Aussagen, welche der Einordnung dienen:

- Guide me!
- Get together!
- Navigation and Routing

Diese Einordnungshilfen zielen darauf ab, von einem aktuellen Ort zu einem anderen zu gelangen. Dazu zählt auch das „Zusammentreffen“ mit einem anderen mobilen Ziel. Der Endpunkt einer Navigation muss dabei eingegeben werden, Treffpunkte können automatisch ermittelt werden.

2.3 OPENGIS

OpenGIS® ist die geschützte Marke, welche den gesamten Standardisierungsprozess für Geodaten verarbeitende Software beschreibt. Es wurde von dem Open Geospatial Consortium® (OGC) geplant und wird durch diese gemeinnützige Organisation weiterhin entwickelt.

Ziel von OpenGIS ist es, Informationen aus raumbezogenen Daten mit Hilfe verschiedener verarbeitender Methoden zu gewinnen. Raumbezogene Daten oder Informationen nennt man die Fakten, die in einer beliebigen Form mit einem bestimmten Ort oder Gebiet in Verbindung gebracht werden können. Dabei kann es sich sowohl um die Koordinaten eines Hauses oder eines Weges, als auch um die Bevölkerungsdichte eines Gebiets handeln. Um Softwarekomponenten zu entwickeln, die dieses Ziel unterstützen und untereinander kompatibel sind, werden im OpenGIS-Prozess standardisierte Datenmodelle und Standard-Schnittstellen spezifiziert (vgl. [And09]). Es soll ermöglicht werden, dass Programme, die OpenGIS-konform programmiert sind, über die Schnittstellen kommunizieren und Daten austauschen können.

2.3.1 Geodaten

Geodaten sind raumbezogene Daten die als Grundlage für alle Geoinformationssysteme dienen. Sie werden in Geodatenbanken oder Dateisystemen gespeichert und verarbeitet. Dabei kann es sich um einfache 2D-Vektordaten oder auch komplexe Geometrien handeln. Es sind ebenfalls georeferenzierte Rasterkarten, XML-basierte Geodaten oder topologische Datenmodelle unter diesem Begriff zusammengefasst. Geodaten werden zur Verarbeitung und Darstellung zwischen den Programmen ausgetauscht. Analyse-, Such- und topologische Verschneidungsfunktionen auf den Daten werden durch die Geodatenbanken zur Verfügung gestellt (vgl. [Bri08]).

Die Datenhaltung kann auf verteilten Datenquellen beruhen. Der Zugriff erfolgt über einem Server, welcher die verschiedenen Datenquellen verwaltet. Den Anwendungsprogrammen stehen so die Geodaten unabhängig der Herkunft in aktueller Qualität zur Verfügung. Zusätzlich ist wie bei jedem Datenbanksystem die Integrität der Daten sicher gestellt. Auf die verschiedenen Abfragemöglichkeiten und die Struktur der Daten wird in den nachfolgenden Kapiteln eingegangen.

Geodaten besitzen nach [Bri08] besondere Eigenschaften, die für die Modellierung von Bedeutung sind. Diese sollen hier kurz vorgestellt werden um den ganzheitlichen Überblick zu wahren:

Geometrische Eigenschaften beschreiben die Lage und Ausdehnung der gespeicherten Objekte. Dies erfolgt über Koordinaten, die einen genauen Punkt definieren. Mit mehreren Punkten oder der Angabe der Ausdehnung werden Flächen repräsentiert.

Topologische Eigenschaften ermöglichen die Beschreibung der räumlichen Beziehung zwischen Objekten. Die wichtigsten sind dabei Nachbarschafts-, Überlappungs- und Inklusionsbeziehungen. Die mathematischen Grundregeln einer Topologie gelten auch hier. Sie ist also zum Beispiel bei Skalierungen und Rotationen bezüglich eines Transformationspunktes unverändert.

Thematische Eigenschaften entsprechen Sachattributen. Darunter zählt man nominale, qualitative und quantitative Eigenschaften. Wenn zu einem Geoobjekt mehrere thematische Attribute existieren, unterscheidet man verschiedene Ebenen dieses Objekts. Ebenen oder auch Layer können unabhängige Daten enthalten, wie zum Beispiel eine Ebene die Bevölkerungsdichte und eine andere die Arbeitslosenquote enthalten kann.

Temporale Eigenschaften bezeichnen den Zeitpunkt oder Zeitraum für den die übrigen Eigenschaften gelten. Die Eigenschaft ist ausschlaggebend, um Veränderungen der Position in Bezug auf die Zeit deutlich zu machen.

Neben den vorgestellten Eigenschaften finden sich zu Geodaten immer auch **Metainformationen**. Metadaten umschreiben immer die eigentlichen Daten und stellen damit zusätzliche Informationen zu dem Datensatz bereit. Bei Geodaten könnten dies das Datenformat, die Genauigkeit oder die Aktualität sein. Sie ermöglichen vor allem eine differenzierte Verarbeitung und Verwendung der Daten.

2.3.2 Geography Markup Language

Geography Markup Language (GML) ist die Auszeichnungssprache, die in OpenGIS zum Austausch und zur Modellierung der Geodaten verwendet wird. GML ist ein „XML-Dialekt" und durch XML-Schemadateien wohldefiniert (vgl. [Bla07], [Bri08]). GML enthält eine Vielzahl von Objekttypen, die die Beschreibung der Geodaten standardisieren. So können in GML neben einfachen (geometrischen) Objekten, sogenannte „Simple Features", auch komplexe Geometrien abgebildet werden.

2.3.3 Features

Unter einem Feature versteht man in OpenGIS die grundlegende Repräsentation eines Objektes. Es dient wie in der objektorientierten Programmierung ein Modell, zur Abbildung eines realen Gegenstandes in die Programmiersprache. Das Feature ist also die Abstraktion eines realweltlichen Phänomens mit all seinen räumlichen Eigenschaften (vgl. [And09]). Da der objektorientierte Ansatz auch hier grundlegend ist, können wir dem Feature eine Objektklasse zuordnen, den sogenannten *Feature Type*.

Feature Types können frei definiert werden und unterliegen in dieser Hinsicht keiner Beschränkung. So kann nach dem Feature-Konzept ein Feature auch aus anderen Features bestehen oder durch eine Geometrie repräsentiert werden. Des Weiteren können sie die unter 2.3.1 beschriebenen Eigenschaften besitzen. ISO 19109 „Geographic information - Rules for application schema." definiert die Regeln für das Erstellen von Anwendungsschemata (siehe [And09], [Int05]) und damit für das zugrunde liegende objektorientierte Datenmodell. In dem ISO 19109 Standard werden die möglichen Elemente eines Feature Types genau beschrieben. Der Standard legt ebenfalls fest, wie die Beziehungen und Vererbungshierarchien im Anwendungsschema angegeben werden können.

Als weiterer wichtiger Standard ist ISO 19107 „Spatial Schema" zu nennen. Er definiert wie die geometrischen Attribute angegeben werden müssen (siehe [And09], [Int03]). Durch die Festlegung der geometrischen und topologischen Klassen ist es möglich, die Geometrie direkt im Feature zu beschreiben. Sie kann aber auch, als Referenz angelegt, auf unabhängige Geometrieobjekte verweisen. Diese Variante ermöglicht auch die Mehrfachnutzung in unterschiedlichen Features.

2.3.4 Web Services

Ein Web Service oder auch Web-Dienst wird für den Datenaustausch zwischen Dienstnutzer und Dienstanbieter verwendet. Die Kommunikation erfolgt mit Hilfe von Webtechnologien wie z.B. dem HTTP-Protokoll. Jeder Service besitzt eine eindeutig definierte Uniform Resource Identifier (URI) und damit eine Schnittstelle um angesprochen werden zu können. Eine solche serviceorientierte Architektur stellt gemeinsam genutzte Funktionalität zur Verfügung.

In Bezug auf OpenGIS haben Services mit der Verarbeitung räumlicher Informationen zu tun. Die OpenGIS-Spezifikationen normieren diese Schnittstellen und ermöglichen verschiedenen Komponenten die einheitliche Kommunikation (vgl. [And11]). Nicht alle OpenGIS-Services sind Web Services, sondern nur eine Teilmenge.

Um einen **OpenGIS-Web Service** (OWS) zu nutzen, muss die anfragende Komponente (der Client) eine HTTP-Request senden. Um die Anfrage zu spezifizieren, können verschiedene Parameter mitgesendet werden. Der Response des Servers ist meist eine Bilddatei oder XML-Datei. Die Kommunikation mit dem Server wird in der Regel „stateless" geführt. Das heißt, dass alle Informationen und Parameter bei jedem Request gesendet werden müssen. Es existieren unterschiedliche Web Services für verschiedene Aufgaben: unter Anderem der **Web Map Service** (WMS), welcher als Response Kartenausschnitte zurück liefert. **Web Coverage Services** (WCS) liefern hingegen Rasterdaten an den Client aus. Nachfolgend wird der **Web Feature Service** (WFS) beschrieben, da er in dieser Arbeit besondere Bedeutung hat.

```
GET http://host[:post]/pfad[?{name[=wert]&}]
```

Code 2.1: OWS Request

Alle OpenGIS-Web Services haben folgende Gemeinsamkeiten:

- Sie sind in der Regel stateless.
- Die bereitgestellten Services stellen ihre Dienste über eine HTTP-Schnittstelle zu Verfügung.
- Von allen OWS wird die Operation *GetCapabilities* zur Verfügung gestellt. Darüber können die Funktionen des jeweiligen Dienstes abgefragt werden. Als Antwort wird ein XML-Dokument generiert und an den Client gesendet.
- GetResourceByID wird ebenfalls von allen OWS definiert. Über diese Funktion wird der Zugriff auf die Metainformationen erlaubt.
- Pflichtparameter jedes Aufrufes sind: **SERVICE**, **REQUEST** und **VERSION**.
- Als Antwort wird vom Server die in der jeweiligen Norm festgelegte Antwort oder eine entsprechende Fehlermeldung gesendet. Fehlermeldungen haben den *MIME-Type text/xml*.
- Die Syntax einer einfachen GET-Anfrage ist im Codebeispiel 2.1 dargestellt. Bei der Kodierung der Anfrage müssen die URL-Spezifikationen (IETF RFC 2396) beachtet werden.

Web Feature Service Wenn die Funktionalitäten eines WMS für die Anwendung nicht ausreichend sind, besteht die Möglichkeit auf WFS zurückzugreifen. Dadurch kann die Software um Analyse und weitere Interpretationsfunktionen erweitert werden. Datenlayer lassen sich nicht mehr nur noch darstellen, sondern dienen dem Extrahieren und Generieren von Informationen. Der Web Feature Service repräsentiert den vom OpenGIS Consortium beschriebenen „Feature Case", bei dem vom Server Geo-Objekte in Form von Vektordaten an den Client gesendet werden. Dieser hat dann die Möglichkeiten der weiteren Verarbeitung und Darstellung (vgl. [Ber05]).

Die dem WFS zugrunde liegende Spezifikation definiert fünf Operationen, die zwei verschiedenen Klassen von WFS zugeordnet werden können. Bei den Klassen handelt es sich erstens um den **Basic WFS**, welcher dem Client lesenden Zugriff auf die Geodaten erlaubt und zweitens den **Transactional WFS**, welcher zusätzlich schreibende Zugriffe ermöglicht. Die Operationen sind folgende:

GetCapabilities Wie bereits im Abschnitt 2.3.4 beschrieben, dient diese Operation dem Auflisten der eigenen Funktionen. Im ausgelieferten XML-Dokument sind die folgenden dienstinstanzbezogenen Metainformationen enthalten:

- Version, Name, Titel, Beschreibung
- Schlüsselwörter
- unterstützte Operationen und Filter
- Feature Types
- Informationen zur Datenbasis
- Verweis auf externe Metadaten (optional)
- Format der Antwort, Transferformate (optional)

DescribeFeatureType Die über GetCapabilities zur Verfügung gestellten Metadaten reichen unter Umständen für eine korrekte Anfrage nicht aus. Aus diesem Grund gibt die Funktion DescribeFeatureType weitere Informationen zur Struktur der Daten zurück. Sie wird nötig, um gezieltere Anfragen und Analysen ausführen zu können. Dazu gehören:

- Properties der Features
- Beziehung zwischen den Features
- Datentypen der einzelnen Properties
- Wertebereich der Datentypen

Die Antwort auf die DescribeFeatureType-Anfrage bildet einen Ausschnitt aus dem Anwendungsschema der Geodaten.

GetFeature Die GetFeature-Operation ermöglicht dem Client die Abfrage von bestimmten Geoobjekten. Die Auslieferung der Features erfolgt in einem GML-Dokument. Mit der Anfrage können Filter und Parameter die Menge der Features eingrenzen und somit gezielte Zugriffe auf den Geodaten ausgeführt werden. Es ist auch möglich einzelne Features per ID auszugeben.

Transaction Transactions werden vom Client genutzt, um die Geodaten zu ändern. Es gibt wie bei jeder Datenbankabfrage die Möglichkeiten: Einfügen von Daten, Updaten bzw. Ändern von Daten und Löschen von Daten. Der Status der Anfrage wird in einem XML-Response ausgegeben.

LockFeature Auf Grund der verteilten Arbeitsweise und mehrfacher gleichzeitiger Zugriffe auf den Datenbestand ist es notwendig, Features während der Bearbeitung zu sperren. Dieses Vorgehen gewährleistet die Konsistenz der Daten und wird mit der Operation LockFeature durchgeführt. Alternativ kann auch die Funktion: **GetFeatureWithLock** verwendet werden. Nach erfolgreicher Transaktion kann die Sperre wieder aufgehoben werden.

```
<ogc:Filter_Capabilities>
  <ogc:Spatial_Capabilities>
    <ogc:Spatial_Operators>
      <ogc:Equals/>
      <ogc:Disjoint/>
      <ogc:Touches/>
      <ogc:Within/>
      <ogc:Overlaps/>
      <ogc:Crosses/>
      <ogc:Intersect/>
      <ogc:Contains/>
      <ogc:DWithin/>
      <ogc:BBOX/>
    </ogc:Spatial_Operators>
  </ogc:Spatial_Capabilities>
  <ogc:Scalar_Capabilities>
    <ogc:Logical_Operators />
    <ogc:Comparison_Operators>
      <ogc:Simple_Comparisons />
      <ogc:Like />
      <ogc:Between />
    </ogc:Comparison_Operators>
  </ogc:Scalar_Capabilities>
</ogc:Filter_Capabilities>
```

Code 2.2: Ausschnitt der aktivierten Filter aus dem GetCapabilities Response

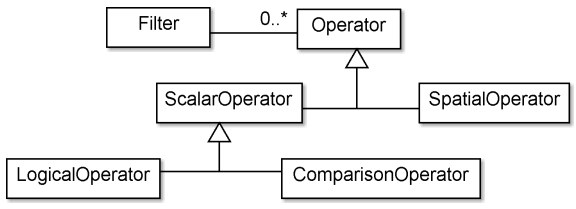


Abbildung 2.2: UML Klassendiagramm des Filter-Schemas

2.3.5 Filter Encoding

In der „OpenGIS-Filter-Encoding-Specification“ (siehe [Ope05]) wird definiert, wie Filterkriterien im XML-Format angelegt und übermittelt werden. Mit Hilfe dieser Filter können die Geodaten tiefgründiger und detaillierter als durch einfache Abfragen analysiert werden. Voraussetzung ist eine wohldefinierte Datenstruktur, welche auf Grund der OpenGIS-Festlegungen bereits gegeben ist (nach [Ber05], [And11]). Grundsätzlich können zwei verschiedene Filtertypen auf die Features angewandt werden: räumliche und Attributfilter (siehe Abbildung 2.2). Welche Filter in der aktuellen Konfiguration des WFS zugelassen sind, lässt sich leicht über die GetCapabilities-Ausgabe erkennen. Im Codebeispiel 2.2 kann man alle definierten Filter in den beiden Unterelementen „Spatial_Capabilities“ und „Scalar_Capabilities“ ablesen.

```
GET http://.../cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS&REQUEST=
    GetFeature&TYPENAME=cities&Filter=<Filter><DWithin><PropertyName>
    Geometry</PropertyName><gml:Point><gml:coordinates
    >135.500000,34.666667</gml:coordinates></gml:Point><Distance units
    ='m'>100</Distance></DWithin></Filter>
```

Code 2.3: Request mit DWithin Filter des WFS

```
<Filter>
  <DWithin>
    <PropertyName>Geometry</PropertyName>
    <gml:Point>
      <gml:coordinates>135.500000,34.666667</gml:coordinates>
    </gml:Point>
    <Distance units='m'>100</Distance>
  </DWithin>
</Filter>
```

Code 2.4: DWithin Filter des WFS in XML-Syntax

Die räumlichen Filter ermöglichen es, geometrische Formen zu erhalten, die sich überschneiden oder zum Beispiel gegenseitig enthalten. Durch weitere Funktionen sind alle geometrischen Beziehungen zwischen zwei Objekten abgedeckt. Der GET-Request im Codebeispiel 2.3 zeigt die Anwendung des DWithin-Filter. Er gibt alle Geodaten als Ergebnismenge zurück, die im Umkreis von 100m der angegebenen Koordinate definiert sind. Da der Filter in XML spezifiziert ist, kann er wie im Codeauszug 2.4 auch übersichtlicher dargestellt werden. Man erkennt deutlich den Filter um den es sich handelt als erstes Element. Im „DWithin“-Filter sind drei Parameter angegeben. Der erste „PropertyName“ ist in jedem Filter vorhanden um zu definieren, auf welche Eigenschaft der Filter angewandt wird. Der zweite Parameter ist der Punkt und der dritte legt den Abstand fest, mit dem gesucht wird.

Die „Scalar_Capabilities“ stellen Attributfilter und logische Filter zur Verfügung. Damit lassen sich einfache Attribute vergleichen und durchsuchen, sowie mehrere Filter und Vergleiche miteinander verknüpfen.

2.4 MAPBIQUITOUS

MapBiquitous ist ein Forschungsprojekt der TU Dresden, welches die Umsetzungsmöglichkeiten für integrierte Indoor und Outdoor-LBS untersucht (vgl. [Keß11]). Es wird der Ansatz eines integrierten Systems für die Verarbeitung von Gebäudedaten und Karten verfolgt. Ziel ist es, eine Anwendung zu erstellen, die existierende Techniken der etablierten Outdoor-LBS mit den Möglichkeiten innerhalb von Gebäuden vereint.

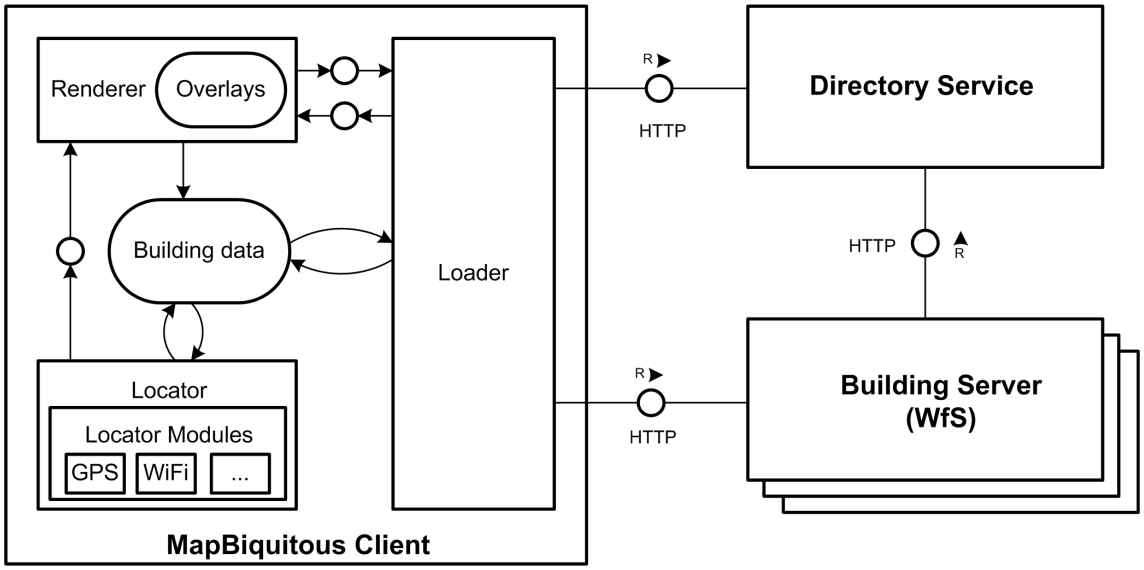


Abbildung 2.3: Architektur des MapBiquitous-Projekts (aus [Spr11])

Grundlage der Anwendung ist eine Client-Server-Architektur, wie sie in Abbildung 2.3 dargestellt ist (siehe [Spr11]). Serverseitig ist die Architektur auf dezentralisierte Gebäudeserver aufgebaut. Sie verwalten jegliche Daten und kommunizieren über Web Feature Services und Directory Services mit dem Client. Alle benötigten Dienste werden vom „MapServer" in der Version 5.2.1 bereitgestellt. Die Datenhaltung erfolgt Standard konform in einer PostGIS-Datenbank (weitere Informationen unter [SS10]). Zu den gespeicherten Daten gehören Informationen zu Geometrien der einzelnen Gebäude sowie deren Etagen. Zusätzlich werden für jede Etage WLAN-Access-Points mit ihren genauen Koordinaten gespeichert. Alle Geodaten verfügen über weitere Metadaten, die zur Navigation und für andere Verarbeitungs- und Darstellungsschritte verwendet werden können.

Die Client-Seite ist wiederum in drei Komponenten unterteilt: den Loader, den Renderer und den Locator. Um die Funktionsweise der Architektur zu verstehen, werden die Hauptkomponenten nachfolgend genauer beschrieben (nach [Spr11]):

Der Locator enthält verschiedene Module, welche die Positionierung gewährleisten. Er ist so konzipiert, dass die Module flexibel erweitert oder ausgetauscht werden können. So ist die Positionsbestimmung mit verschiedenen Technologien immer an die aktuelle Verfügbarkeit von Sensoren und Accesspoints angepasst. Die aktuelle Konfiguration ist so angelegt, dass die Positionierung per GPS oder WLAN funktioniert. Verfahren wie RFID und weitere Sensoren aktueller

Smartphones befinden sich im Untersuchungsstatus.

Der Renderer empfängt die stetigen Aktualisierungen der Position. Die Komponente wertet die empfangenen Änderungen aus und fordert bei Bedarf beim Loader weitere Geodaten an. Beim Darstellen der Daten wird das Konzept der Overlays verfolgt. Es werden die Geometrien als weitere Schicht über der Karte gezeichnet. Verschiedene Detailstufen werden in unterschiedlichen Layern repräsentiert. Sie können einzeln oder überdeckend gerendert werden, wobei immer die oberste Schicht für den Nutzer zuerst sichtbar ist.

Der Loader übernimmt die Kommunikation mit den Gebäudeservern. Sobald eine Anfrage des Renderers nach weiteren Gebäudedaten vorliegt - zum Beispiel nach dem Laden einer weiteren Detailschicht - lädt die Komponente dynamisch die geforderten Daten nach. Der Loader führt die Requests an den Server aus und verarbeitet den Response. Wenn die Anfrage ohne Fehler beantwortet wird, werden die erhaltenen XML-Daten geparsed und in das Ortsmodell überführt. Sie stehen dann dem Renderer zur Darstellung zur Verfügung.

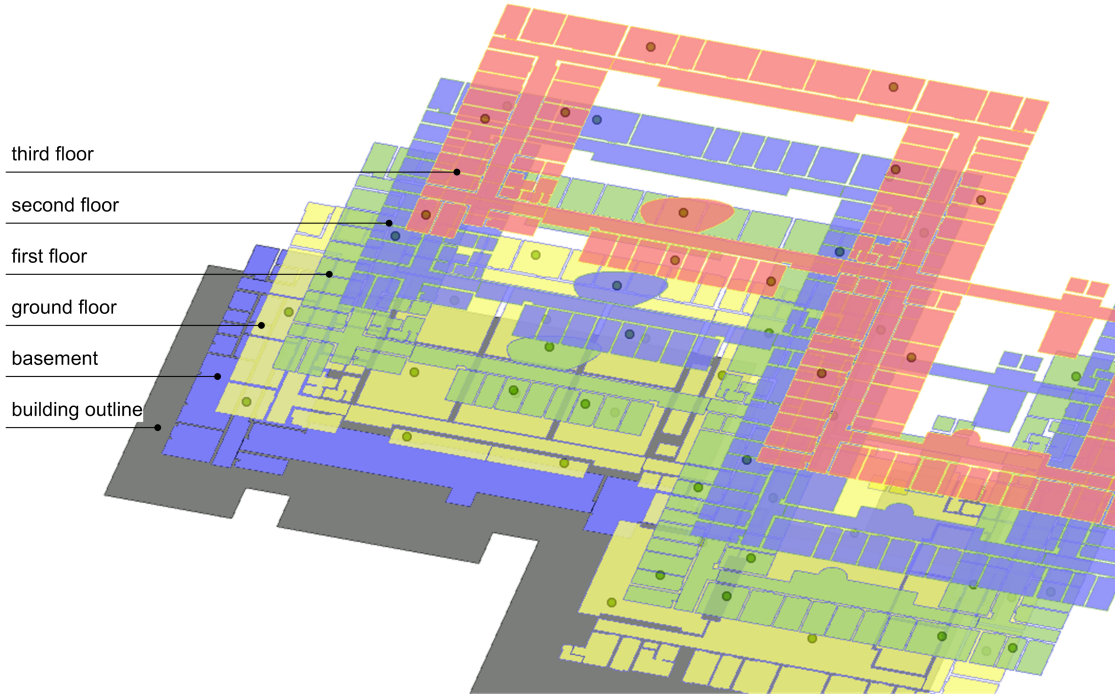


Abbildung 2.4: Layermodell der Fakultät Informatik TU Dresden (aus [Spr11])

Das Ziel der Client-Anwendung ist es, eine nutzerfreundliche Oberfläche zu schaffen, welche die vom Gebäudeserver erhaltenen Daten in ansprechender Weise präsentiert. Über das Userinterface werden ebenfalls zur Verfügung gestellte Funktionen des Gebäudeservers für den Nutzer verwendbar gemacht. In Abbildung 2.4 sind alle möglichen Layer am Beispiel der Fakultät Informatik dargestellt. Die unterste Schicht, das *building outline*, stellt den Grundriss dar und wird geladen, sobald der Grundriss für den Nutzer sichtbar wird. Für den Fall das weitere Details des Gebäudes angefordert werden, muss der Nutzer sich für die entsprechende Etage entscheiden. Sie steht dann als weiteres Layer zur Verfügung. Um Unkenntlichkeiten zu vermeiden, wird im-

mer nur eines der Layer über der Karte geladen. Die Etagen stellen den höchsten Detailgrad - das Level of Detail 3 (LOD3) - dar. Hier können die Räume angewählt werden um Zusatzinformationen zu erhalten.

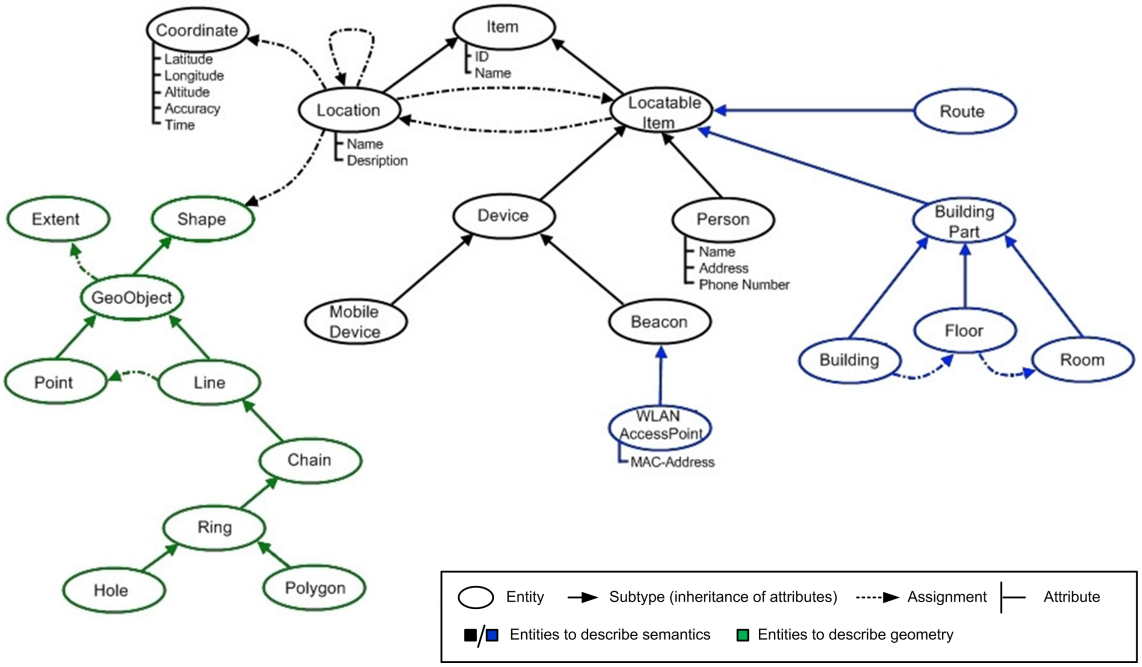


Abbildung 2.5: Ortsmodell des MapBiquitous-Projekts (aus [Sch09a])

Das LOD3 schöpft die meisten Möglichkeiten des Ortsmodells (Abbildung 2.5) nach [Sch09b] und [Sch09a] aus. Im Modell sind alle Geometrien und möglichen Gebäudeteile enthalten. Jede dargestellte Geometrie erbt von *Item*. Sie hat so zusätzlich alle geometrischen Attribute eines *GeoObjects*. Die Geometrieeigenschaften ermöglichen das korrekte Einzeichnen auf der Karte. Die Unterklassen, speziell von *LocatableItem*, ermöglichen es, Metadaten zur jeweiligen Geometrie zu speichern. Diese Metadaten enthalten Zusatzinformationen zur Nutzung oder Raum- und Gebäudebezeichnung.

Das Userinterface (UI) ist mit einer zentralen Kartenansicht einfach gehalten. Um Daten nachzuladen sind immer Nutzerinteraktionen notwendig. So werden die Etagenpläne erst geladen, wenn der Nutzer den entsprechenden Grundriss berührt und im folgenden Pop-Up-Menü die Etage wählt. Durch erneutes Berühren der Räume erscheint ein ähnliches Pop-Up-Menü aus dem man weitere Informationen erhält oder zur Etagenwahl navigieren kann. Über dieses Menü lässt sich ähnlich der Etagenwahl eine Liste aller verfügbarer Räume anzeigen und auswählen. Gewählte Räume werden farblich hervorgehoben. Zoomen ist in der Kartenansicht über zwei Buttons („+“ zum Vergrößern des Kartenausschnittes und „-“ zum Verkleinern) oder intuitiv über das Spreizen oder Zusammenführen zweier Finger möglich.

2.5 ZUSAMMENFASSUNG

Ortsbezogene Dienste erfreuen sich auf vielen Plattformen immer größerer Beliebtheit. Von der anfänglichen Konzeption zur Unterstützung der Rettungsdienste haben sie sich vielfältig weiterentwickelt und stehen jedem Endnutzer zur Verfügung. Die aufgezeigten Fortschritte vollzogen sich bisher weitestgehend im Outdoor-Bereich. Es wurden dazu Anwendungsfälle aus den unterschiedlichsten Sektoren vorgestellt, die verdeutlichen, dass Location-Based Services sowohl für Unternehmen als auch für private Anwender Nutzungsvorteile bereitstellen.

Die Aufgabe dieser Arbeit ist es, mit den etablierten OpenGIS-Standards eine Integration von vorhandenen Outdoor-Services und neu zu konzipierenden Indoor-Services herzustellen. Das MapBiquitous-Projekt dient dabei als Ausgangspunkt. Es stellt bereits einfache Funktionen zum Laden und Darstellen von Gebäudedaten zur Verfügung. Die Voraussetzungen für erweiterte und integrierte Outdoor/Indoor-LBS sind noch nicht gegeben. Ebenfalls wird ein flexibles Framework als Grundlage für eine kontinuierliche Weiterentwicklung und zukünftige Forschungsschwerpunkte benötigt.

3 VERWANDTE ARBEITEN

Im folgenden Kapitel werden wichtige vorhandene Arbeiten untersucht und marktführende Lösungen analysiert. Dabei sollen die jeweils innovativsten Lösungsansätze hervorgehoben werden, um sie abschließend miteinander vergleichen und bewerten zu können. Auf gleichwertige Umsetzungen wird nicht doppelt eingegangen, damit am Ende des Kapitels ein umfassender Überblick der verschiedensten aktuellen Systeme und deren Funktionen gewährleistet ist.

3.1 LOCATION-BASED SERVICES AM BEISPIEL EINES LOKALFÜHRERS

In der Arbeit von Komenda [Kom08] wurden die sich neu eröffnenden Möglichkeiten der zunehmenden Mobilität eines Nutzers untersucht. Verschiedene Strategien und Techniken in Verbindung mit LBS konnten vorgestellt werden, um letztendlich daraus ein Konzept für einen Lokalführer zu entwickeln. Der gesetzte Schwerpunkt lag dabei auf der innovativen Umsetzung der Software und der damit verbundenen Verknüpfung von herkömmlichen Internetdiensten und den neu ergründeten mobilen Möglichkeiten.

Entstanden ist eine umfangreiche Plattform zur Erstellung, Bewertung und Lokalisierung von Restaurants im weitesten Sinne. Mit Hilfe dieses Systems werden dem mobilen Nutzer alle Möglichkeiten zum Suchen und Finden des nächsten oder eines bestimmten Restaurants gegeben. Auf Kernfunktionalitäten und genaue Strukturierung der Anwendung wird in den nächsten Absätzen eingegangen.

3.1.1 Lokaldetails

Alle aufgenommenen Lokale werden jeweils durch verschiedene Informationen repräsentiert. Diese Informationen dienen zur Beschreibung eines Lokals und um effektiv nach den Lokalen

suchen und sie filtern zu können. Aus vielen evaluierten Details haben sich die folgenden als wichtigste für die Umsetzung der Software herausgestellt:

- Name
- Kategorie
- Küche
- Öffnungszeiten
- Adresse
- Preisniveau
- Extras
- Bewertungen
- Foto
- Stadtplan

Anhand dieser aussagekräftigen Eigenschaften ist jedes Lokal umfassend beschrieben. Auf die Kategorie und die Bewertungen wird näher eingegangen. Die übrigen Eigenschaften sind selbsterklärend und drücken jeweils eine exakt zu beschreibende Lokaleigenschaft aus. So ist im Namen die Bezeichnung festgelegt, unter Foto findet sich ein ausdrucksstarkes Foto des Gastraumes oder der Lokalität.

Kategorieeinteilung Bei der Einteilung wurde darauf geachtet, dass alle möglichen Kategorien aller Lokale abgedeckt werden. Hiermit wird also die Art des Lokals zum Beispiel *Restaurant* angegeben. Zusätzlich wurde damit die Länge der Auswahlliste auf eine ausreichende Anzahl von Einträgen eingegrenzt. Die Übersichtlichkeit bleibt dadurch gewahrt. Ein weiterer Nebeneffekt ist, dass der Redakteur, welcher das Lokal anlegt, die Auswahl genau überdenken muss. Die Auswahl der Kategorien ist allgemeingültig und für alle „Geschäfte" passend wählbar und somit für jeden Eintrag wiederverwendbar. Die Liste der Kategorien muss daher zum späteren Zeitpunkt und mit zunehmender Anzahl an Einträgen nicht angepasst werden.

Bewertungen Jedes Lokal kann von allen registrierten Benutzern bewertet werden. Es wird eine Bewertungsskala von der niedrigsten (1) bis zur besten Bewertung (5) bereitgestellt. Beim Anzeigen der Lokaleigenschaften werden die Bewertungen aller Nutzer gemittelt. Die Tatsache, dass die Nutzer registriert sein müssen, verhindert Spam und schränkt gravierend falsche Angaben ein. Es werden dadurch auch Mehrfachbewertungen verhindert. Diese Konzeption gibt dem Nutzer die Möglichkeit, seine Bewertung bei einem späteren Besuch anzupassen.

3.1.2 Wichtige Funktionen

Die Anwendung verfügt über verschiedene Funktionen, um die Ergebnisse zu suchen und mit Filtern einzugrenzen. Am Anfang aber steht die Möglichkeit ein Lokal anzulegen, welche im nächsten Abschnitt erläutert wird. Darauf folgend wird noch auf weiteren interessanten und nützlichen Funktionsumfang eingegangen.

Lokale eintragen Ebenso wie das Bewerten, ist das Eintragen von Lokalen auf einen bestimmten Nutzerkreis beschränkt, zum Beispiel auf Administratoren. Sie können auch Änderungen vornehmen. Beim Eintragen können alle Lokaldetails (siehe 3.1.1) berücksichtigt und entsprechend festgelegt werden. Vollständigkeit ist bei allen Einträgen grundsätzlich erwünscht aber keine Voraussetzung für das Erstellen einer Lokalität. Dazu sind lediglich die wichtigsten Details wie „Name", „Kategorie" und „Adresse" notwendig.

Listen und Favoriten Die analysierte Anwendung stellt dem Nutzer zwei verschiedene Möglichkeiten für die Verwaltung bekannter Lokale zur Verfügung. Einerseits existiert eine History in der alle zuletzt betrachteten Lokale aufgelistet werden. So kann man schnell von einem Restaurant zum vorherigen springen. Diese Liste wird automatisch ohne Zutun des Nutzers angelegt. Andererseits kann der Nutzer eine Favoritenliste selber pflegen. Dazu kann er beliebte Lokale „merken", diese werden erst dann zu den Favoriten hinzugefügt.

3.1.3 Fazit

Der Zugriff auf die bereit gestellten Informationen erfolgt einerseits über eine für den Nutzer komfortable Freitextsuche, welche zu jedem gefundenen Ort die selben Informationen (3.1.1) bereitstellt. Das macht die Ergebnisse optimal vergleichbar und für den Nutzer einschätzbar. Der Entwickler kann hingegen immer auf die selben Eigenschaften zugreifen und diese entsprechend übersichtlich angeordnet darstellen. Die einheitliche Visualisierung der Ergebnisse ist somit ebenfalls gewährleistet. Ein anderer Vorteil des fest definierten Umfangs an Metainformationen pro Ort, ist die Nutzungsmöglichkeit von Filtern. Sie ermöglichen schnelles Eingrenzen und Durchsuchen der Ergebnisse sowie eine übersichtliche Datenstruktur.

Die Schnittstellen, die zur Datenabfrage und zum Eintragen der Daten genutzt werden, sind sehr einfach strukturiert. Die Abfrage bzw. Suche von Orten wird über einen GET-Request mit entsprechenden Suchparametern gewährleistet. Der Server antwortet mit den im Response enthaltenen Orten und ihren jeweiligen Details. Eine weitere Schnittstelle ist zum Einfügen und Aktualisieren von Orten notwendig. Dieser POST-Request muss sowohl die Authentifizierungsdaten des Nutzers, die zu aktualisierende oder erstellte Lokalität und die zugehörigen Details enthalten. Sofern der Nutzer die benötigten Rechte besitzt und die Validierung der Details erfolgreich verläuft, wird der „success" im Response gesendet.

3.2 GOOGLE MAPS

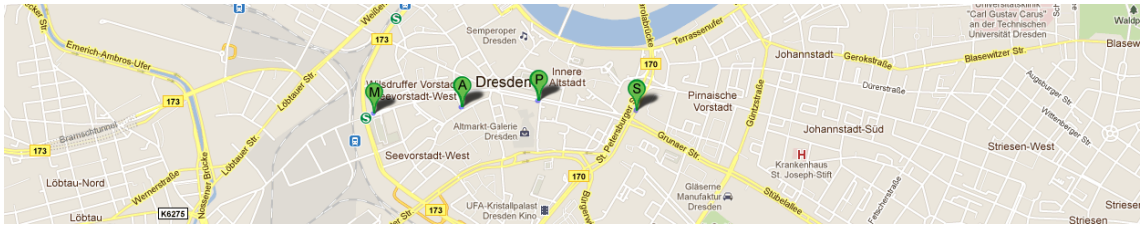
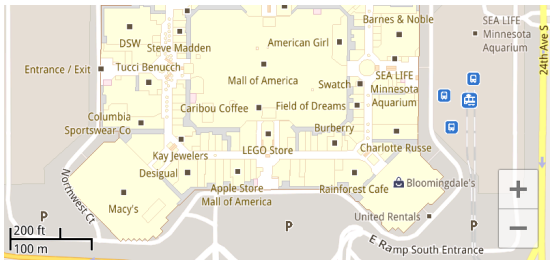


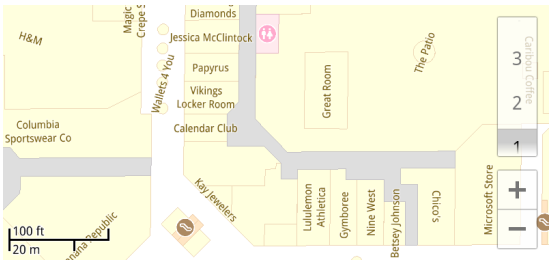
Abbildung 3.1: Google Maps Desktopversion, Kartenausschnitt

Google Maps ist einer der derzeitig größten Kartendienstleister. Für den Nutzer werden auf der Kartenoberfläche Informationen angezeigt und mit Koordinaten verknüpft. Dazu wird Google Maps auf den meisten mobilen Endgeräten als eigenständige Applikation angeboten. Aber auch im Browser stehen alle Funktionen und Karten für den Desktopnutzer zur Verfügung. Um Daten auch in anderen Anwendungen darstellen zu können, hat Google eine umfangreiche API erarbeitet und veröffentlicht. In diesem Kapitel geht es darum, die Funktionen von Google Maps in den drei eben benannten Teilbereichen zu analysieren.

Die Android App in der Version 6.0.3 stellt dem Nutzer zahlreiche Funktionen zur Verfügung. Diese reichen vom Auffinden eines Ortes bis zur punktgenauen Navigation zu einer bestimmten Adresse. Dabei werden die nativen Standort- und Kommunikationsfeatures des jeweiligen Smartphones vollständig ausgeschöpft. Um einen Überblick über den Funktionsumfang zu erhalten, werden diese hier kurz erläutert. Ausgangssituation für die meisten der Funktionen ist, dass die App annähernd den aktuellen Standort ermitteln kann.



(a) Kartenausschnitt mit Indoor-Informationen



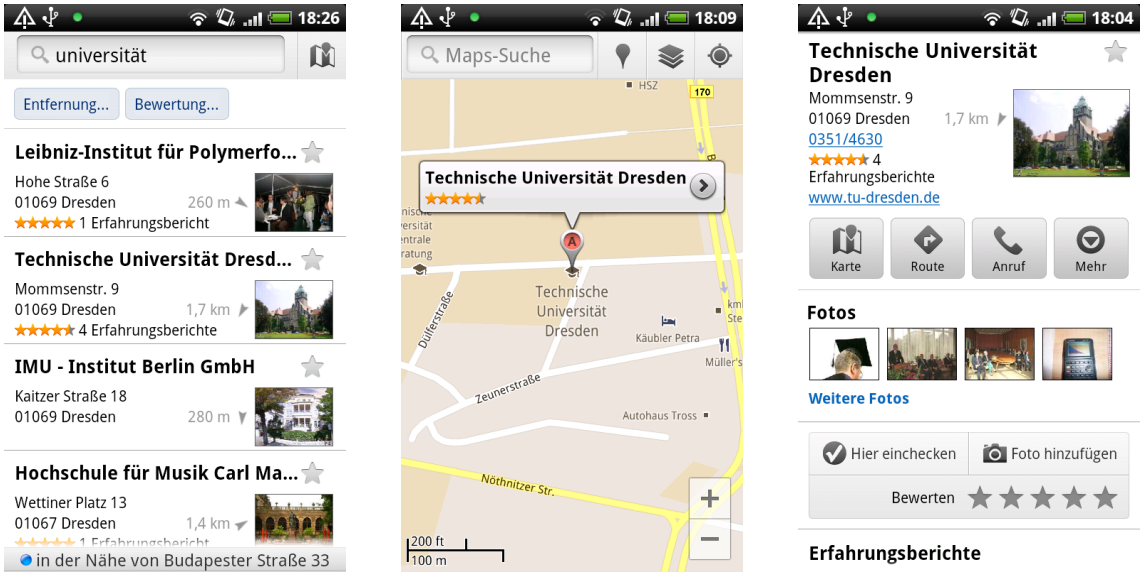
(b) Indoor-Karte der American Mall

Abbildung 3.2: zwei Kartenausschnitte mit verschiedenem Zoomlevel auf Indoor-Kartierungen

Ab Version 6.0.0 wurde der App ein neues Feature hinzugefügt. In Kooperation mit „American Mall", „IKEA" und einigen Flughäfen hat Google in den USA die Innenräume der Gebäude kartiert und für die Nutzer der App freigeschalten[1] [2]. Ab einem nicht näher definiertem Zoomlevel, in dem das Gebäude deutlich identifizierbar ist, werden Indoor-fähige Gebäude in einer anderen Schattierung dargestellt (siehe Abbildung 3.2a). Vergrößert der Nutzer den Ausschnitt weiter, erhält er automatisch die Ansicht des Erdgeschosses mit allen eingetragenen Räumen (Abbildung 3.2b). Über den Buttons zum Zoomen, wird eine Wahlleiste für die Etagen eingeblendet. Wie

[1] `http://googleblog.blogspot.com/2011/11/new-frontier-for-google-maps-mapping.html` (Zugriff am 06.01.2012)

[2] `http://maps.google.com/help/maps/starthere/index.html` (Zugriff am 06.01.2012)



(a) Suchergebnisse

(b) Kartenausschnitt mit einem Ergebnis

(c) Detailansicht

Abbildung 3.3: drei Screenshots der Google Maps Android App

im Screenshot 3.2b zu sehen ist, kann der Nutzer in der Etagenwahl scrollen, wenn mehr Etagen als verfügbarer Platz vorhanden sind. Im Indoor-Bereich ist das Anwählen der einzelnen Räume nicht möglich. Ebenfalls ist eine Navigation zwischen Räumen nicht implementiert. Für die Kartierung weiterer Gebäude existiert ein Workflow, bei dem digitalisierte Etagenpläne in Google Maps automatisiert eingepasst werden. Anschließend werden Räume erkannt und es können Informationen hinzugefügt werden.

3.2.1 Android App

Die App verfügt über unterschiedliche Funktionen und Ansichten um dem Nutzer Informationen darzustellen. Die wichtigsten werden nachfolgend erläutert:

Suche dabei werden passende Suchtreffer in der Umgebung mit Markern angezeigt (siehe Screenshot 3.3b). Wenn zu viele Suchtreffer im Kartenausschnitt dargestellt werden müssen, werden nur die relevantesten mit Markern versehen. Die anderen sind durch einen Punkt markiert.

In der Nähe suchen findet ausschließlich Suchergebnisse in einem festgelegten Radius, wobei die Suche auch weiter entfernte Ergebnisse auflistet.

Was ist in der Nähe? ist hingegen eine Anfrage, die eingetragene konkrete Orte (keine Adressen) mit einem maximalen Abstand zum aktuellen Standort auflistet und darstellt.

Kurzinformationen werden in der Form einer Sprechblase für verschiedene Orte angezeigt (siehe Screenshot 3.3b). Diese Orte können durch langes Berühren des Bildschirms geöffnet werden, sie können aber auch Marker einer Suche oder Wegbeschreibung sein. Die angezeigte

Kurzinformation enthält maximal den Namen / die Adresse des Ortes, eine Bewertung (in Form von Sternen), ein Foto und eine Kurzbeschreibung. Die Kurzbeschreibung wird aber auch in der Auflistung der Suchergebnisse, wie im Screenshot 3.3a gezeigt, verwendet.

ausführliche Informationen erhält man beim Klick auf eine Kurzinformation. Sie beinhaltet eine Vielzahl weiterer beschreibender Eigenschaften (siehe Screenshot 3.3c). Dazu zählen beispielsweise Nutzerkommentare, Öffnungszeiten, weitere Fotos, Abfahrtzeiten, genaue Adresse, Telefonnummern, weitere Interaktionsmöglichkeiten. Zusätzlich werden „Verwandte Orte" also weitere mögliche Suchtreffer angezeigt.

Navigation ermöglicht das Navigieren vom aktuellen Standort aus oder das Berechnen einer Route zwischen zwei verschiedenen Punkten.

3.2.2 Desktopversion

Die Desktopversion unterscheidet sich funktional nicht von der mobilen App. Das Bedienkonzept, das Layout der Karten und angezeigten Suchergebnisse differenziert sich auf Grund des verfügbaren Platzes aber sehr. Zu jedem aktuellen Kartenausschnitt werden nützliche Hinweise neben der Karte eingeblendet ohne diese zu beeinflussen. Die Ermittlung des Nutzerstandortes fällt technologisch bedingt weg. Ihn kann der Nutzer aber bei Umgebungssuchen und Routenberechnungen selber angeben.

3.2.3 Developer API

Grundsätzlich gliedert sich die API in drei Bereiche. Die Bereiche oder besser Frameworks stellen dem Programmierer unterschiedliche Funktionalitäten zur Verfügung. Sie können unabhängig von einander verwendet werden, sind aber auch kombinierbar. Prinzipiell werden alle Komponenten für verschiedene Entwicklungssprachen bereitgestellt (JavaScript, Flash, Java u.a.), ohne Berücksichtigung der Programmiersprache werden in den nachfolgenden Abschnitten die Schnittstellen und Möglichkeiten beschrieben.

Google Data Protocol Dieses von Google entwickelte Protokoll beschreibt die Grundlagen, die in den folgenden API's Anwendung finden[3]. Dazu gehören hauptsächlich: Abfragen, Einfügen, Ändern und Löschen von Einträgen. Diese Einträge können verschiedene Objekte sein. Möglich wären zum Beispiel: Fotos, Einträge zu Lieblingsrestaurants, Zwischenstop

```
GET /myFeed
```

Code 3.1: Google Data Protocol: Request nach der Ressource „myFeed"

```
200 OK

<?xml version='1.0' encoding='utf-8'?>
```

[3] `http://code.google.com/intl/de-DE/apis/gdata/docs/2.0/basics.html` (Zugriff am 29.10.2011)

```
<feed xmlns='http://www.w3.org/2005/Atom'
      xmlns:gd='http://schemas.google.com/g/2005'
      gd:etag='W/"C0QBRXcycSp7ImA9WxRVFUk."'>
  <title>Foo</title>
  <updated>2006-01-23T16:25:00-08:00</updated>
  <id>http://www.example.com/myFeed</id>
  <author>
    <name>Jo March</name>
  </author>
  <link href='/myFeed' rel='self'/>
</feed>
```

Code 3.2: Google Data Protocol: Response für den Request 3.1

So wird im Codeausschnitt 3.1 eine Ressource mit Hilfe der bekannten URI abgerufen. Dazu wird ein HTTP-GET-Request an den Server gesendet. Der Server wiederum antwortet mit der abgefragten Ressource (siehe 3.2). Dieser Response enthält den Status *200 OK* und gibt damit an, dass kein Fehler während der Abfrage aufgetreten ist. Des Weiteren enthält er die Metadaten der Ressource. Hier ist zu beachten, dass keine Einträge in der Ressource gespeichert sind. Es werden daher nur allgemeine Daten wie „author", „title" oder Informationen zur Versionierung (hier „gd:etag" und Datum des letzten Updates) zurück geliefert.

POST /myFeed

```
<?xml version='1.0' encoding='utf-8'?>
<entry xmlns='http://www.w3.org/2005/Atom'>
  <author>
    <name>Elizabeth Bennet</name>
    <email>liz@gmail.com</email>
  </author>
  <title type='text'>Entry 1</title>
  <content type='text'>This is my entry</content>
</entry>
```

Code 3.3: Google Data Protocol: Einfügen eines Eintrags

Ein wichtiger Aspekt ist die Datengewinnung und damit die Notwendigkeit, Daten in eine Ressource einzufügen. Im Codebeispiel 3.3 wird ein Eintrag mit den Werten „author", „title" und „content" in die Ressource „myFeed" mittels *POST* Request eingefügt. Verläuft die Anfrage erfolgreich, meldet der Server *201 CREATED* und sendet gleichzeitig die erstellten Daten mit. Diese Daten sind um die bereits besprochenen Versionierungsinformationen erweitert. Sie wurden vom Server generiert und können beim Einfügen nicht explizit angegeben werden.

GET /myFeed?q=This

Code 3.4: Google Data Protocol: Freitextsuche in der Ressource „myFeed"

```
200 OK

<?xml version='1.0' encoding='utf-8'?>
<feed xmlns='http://www.w3.org/2005/Atom'
    xmlns:gd='http://schemas.google.com/g/2005'
    gd:etag='W/"S0wCTlpIlip7ImA0X0QI."'>
  <title>Foo</title>
  <updated>2006-01-23T16:26:03-08:00</updated>
  <id>http://www.example.com/myFeed</id>
  <author>
    <name>Jo March</name>
  </author>
  <link href='/myFeed' rel='self'/>
  <entry gd:etag='"CUUEQX47eCp7ImA9WxRVEkQ."'>
    <id>http://www.example.com/id/1</id>
    <link rel='edit' href='http://example.com/myFeed/1/'/>
    <updated>2006-01-23T16:26:03-08:00</updated>
    <author>
      <name>Elizabeth Bennet</name>
      <email>liz@gmail.com</email>
    </author>
    <title type='text'>Entry 1</title>
    <content type='text'>This is my entry</content>
  </entry>
</feed>
```

Code 3.5: Google Data Protocol: Ergebnis der Suche 3.4 mit einem Ergebnis

Um unbekannte Einträge und Ressourcen aufrufen zu können, wird eine Suche benötigt. Da die genaue Adresse der Ressource nicht bekannt ist, kann die Abfrage nicht analog dem Beispiel 3.1 aufgerufen werden. Die einfachste Möglichkeit der Suche nach einem Stichwort wird im Code 3.4 exemplarisch gezeigt. Die Ressource wird mit dem Parameter *q=This* angefragt. Das bedeutet, dass alle Einträge, die „This“ enthalten als Ergebnismenge zurück geliefert werden. Die entsprechende Ergebnismenge mit einem passenden Eintrag ist im Codebeispiel 3.5 zu sehen. Neben den Metadaten der Ressource, werden im XML-Element „entry“ die Eigenschaften und Attribute des Eintrags aufgelistet.

Google Places

Das Google Places API ist ein Dienst, der mittels HTTP-Anfragen Informationen über Orte zurück gibt. Ein „Ort“ ist im Kontext dieses API's definiert als eine Einrichtung, ein geografischer Standort oder ein besonders interessanter Punkt. Bei Ortsanfragen werden Standorte anhand von Breitengrad/Längengrad-Koordinaten angegeben.[4]

[4] `http://code.google.com/intl/de-DE/apis/maps/documentation/places/` (Zugriff am 29.10.2011)

```
GET http://maps.google.com/maps/api/place/search/xml?location
    =40.717859,-73.957790&radius=1600&client=clientId&sensor=
    true_or_false&signature=SIGNATURE
```

Code 3.6: Google Places: Ortssuchanfrage

```
<PlaceSearchResponse>
 <status>OK</status>
 <result>
  <name>Williamsburg</name>
  <type>locality</type>
  <type>political</type>
  <icon>http://maps.gstatic.com/mapfiles/place_api/icons/geocode-71.png
      </icon>
  <reference>CkRAAAAAUhZG...Yy0b4-sd1zCUu9P8</reference>
 </result>
 <result>
  ...Attribute...
 </result>
  ...weitere results...
</PlaceSearchResponse>
```

Code 3.7: Google Places: Antwort auf 3.6

```
GET http://maps.google.com/maps/api/place/details/xml?reference=
    REFERENCE_ID&client=clientId&sensor=true_or_false&signature=
    SIGNATURE
```

Code 3.8: Google Places: Ortsdetailanfragen

```
<PlaceDetailsResponse>
 <status>OK</status>
 <result>
  <name>Peter Luger Steakhouse</name>
  <vicinity>Broadway, Brooklyn</vicinity>
  <type>restaurant</type>
  <type>food</type>
  <formatted_address>178 Broadway, Brooklyn, New York 11211, United
      States</formatted_address>
  <address_component>
   <long_name>178</long_name>
   <short_name>178</short_name>
   <type>street_number</type>
  </address_component>
  <address_component>
   <long_name>Broadway</long_name>
   <short_name>Broadway</short_name>
```

```
    <type>route</type>
   </address_component>
   ...weitere address components...
   <geometry>
    <location>
     <lat>40.7099090</lat>
     <lng>-73.9625800</lng>
    </location>
   </geometry>
   <rating>3.7</rating>
   <url>http://maps.google.com/maps/place?cid=11421630699047551449</url>
   <icon>http://maps.gstatic.com/mapfiles/place_api/icons/restaurant-71.
       png</icon>
   <reference>CkQ6AAAAd...4oT4Sof9RHoGD4</reference>
   <id>ddf9cee27322c3bf</id>
  </result>
</PlaceDetailsResponse>
```

Code 3.9: Google Places: Antwort auf 3.8

Es sind dabei zwei unterschiedliche Anfragen möglich. **Ortssuchanfragen** mit den Parametern: „location", „radius", „sensor", „client", „signatur" (siehe GET-Request 3.6) und die sogenannten **Ortsdetailanfragen** mit der eindeutigen „reference" eines Ortes (siehe GET-Request 3.8). Als Antwort auf Ortssuchanfragen erhält man maximal 20 Ergebnisse, die die gefundenen Elemente repräsentieren, um weitere Anfragen zu ermöglichen. So werden wie im Codeausschnitt 3.7 dargestellt, nur die wichtigsten Metadaten eines Ergebnisses zurückgeliefert. Der Response einer Detailanfrage enthält Details, wie genaue Adresse, Telefonnummer und „rating" des Ortes. Zusätzlich werden Typen und nahe gelegene Orte pro Ergebnis angegeben, um die Einschränkung und Verfeinerung der Suche zu ermöglichen. Der Codeauszug 3.9 zeigt am Beispiel der „address_component" deutlich, dass die Detailtiefe bei gut gepflegten Daten sehr hoch sein kann.

```
GET http://maps.google.com/maps/api/geocode/xml?address=1600+
    Amphitheatre+Parkway,+Mountain+View,+CA&sensor=true_or_false
```

Code 3.10: Google Geocoding: Adressanfrage

Google Geocoding

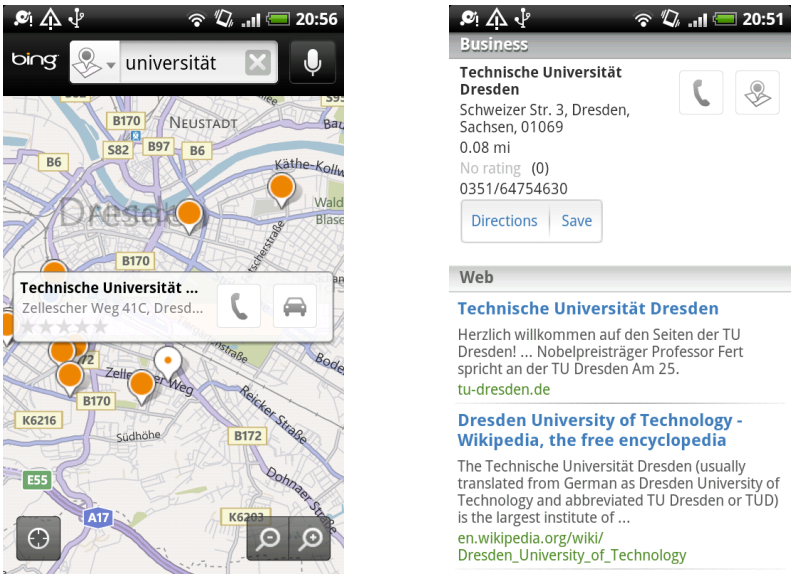
> Die Geocodierung ist der Vorgang der Konvertierung von Adressen (z. B. „1600 Amphitheatre Parkway, Mountain View, CA" siehe 3.10) in geografische Koordinaten (z. B. geografische Breite 37.423021 und geografische Länge -122.083739)[...]. Zusätzlich können Sie mit dem Service den Umkehrprozess (die Umwandlung von Koordinaten in Adressen) ausführen.[5]

[5] `http://code.google.com/intl/de-DE/apis/maps/documentation/geocoding/` (Zugriff am 29.10.2011)

Geocoding hat damit die Funktion nach einer vorgegebenen Adresse zu suchen, dabei werden detaillierte Adressinformationen und genaue Koordinaten als Ergebnis zurückgeliefert. Der Response entspricht dem aus Beispiel 3.8 mit der Einschränkung, dass nur „type“, „formated_adress“, „adress_components“ und „geometry“ Attribute zurückgeliefert werden. In umgekehrter Richtung erhält man für die Eingabe der Koordinaten die verortete Adresse.

3.3 BING MAPS

Als weiterer Anbieter von LBS wird in diesem Kapitel auf die Innovationen von Bing Maps eingegangen. Bing ist die von Microsoft entwickelte Internet-Suchmaschine. Ähnlich Google stellt Bing seinen Nutzern eine Desktopvariante und eine Application für mobile Endgeräte zur Verfügung. Hier ist aber zu beachten, dass der Fokus der aktuellsten Entwicklungen ganz klar auf „Windows Phone“ liegt. Dementsprechend sind auch die zur Verfügung gestellten Developer API's für „Windows Phone“-Entwickler signifikant besser dokumentiert.



(a) Kartenausschnitt mit mehreren Ergebnissen

(b) Detailansicht mit weiteren Suchergebnissen

Abbildung 3.4: zwei Screenshots der Bing Maps Android App

3.3.1 Android App

Nichts desto trotz existiert eine von Microsoft entwickelte Android App in der Version 1.01.309. Sie umfasst neben allen Kartenanwendungen auch die „normalen“ Funktionen der Bing-Suche. Die Funktionen, die unter 3.2.1 beschrieben wurden, sind in komplettem Umfang auch bei Bing Maps zu finden. Die Benennung ist teilweise abweichend, die Ergebnisse der einzelnen Suchanfragen und die aufgelisteten Details eines Ortes unterscheiden sich jedoch, wie in den Screenshots 3.4 erkennbar ist, kaum.

Zusatzfunktionen, die dem Nutzer ein effektiveres Interagieren ermöglichen sind aber deutlich erkennbar. So kann der User schon bei angezeigter „Kurzinformation“ auf ein Telefonie- oder Navigationssymbol „klicken“ (Abbildung 3.4a) und erspart sich damit den Umweg über die Detailansicht des jeweiligen Ergebnisses.

3.3.2 Developer API

Auch die Schnittstellen für den Entwickler unterscheiden sich wenig in ihrer Grundfunktion von denen der Google API's 3.2.3. Es gibt nichtsdestotrotz einige erwähnenswerte Unterschiede.

```
<Location>
  <Name>1 Microsoft Way, Redmond, WA 98052</Name>
  <BoundingBox>
    <SouthLatitude>47.636705672917948</SouthLatitude>
    <WestLongitude>-122.137016420622</WestLongitude>
    <NorthLatitude>47.6444311080593</NorthLatitude>
    <EastLongitude>-122.1217297861384</EastLongitude>
  </BoundingBox>
  <EntityType>Address</EntityType>
  <Address>
    ... Adressdetails ...
  </Address>
  <Confidence>Medium</Confidence>
  <MatchCode>Good</<MatchCode>
  <GeocodePoint>
    ... Koordinaten ...
  </GeocodePoint>
</Location>
```

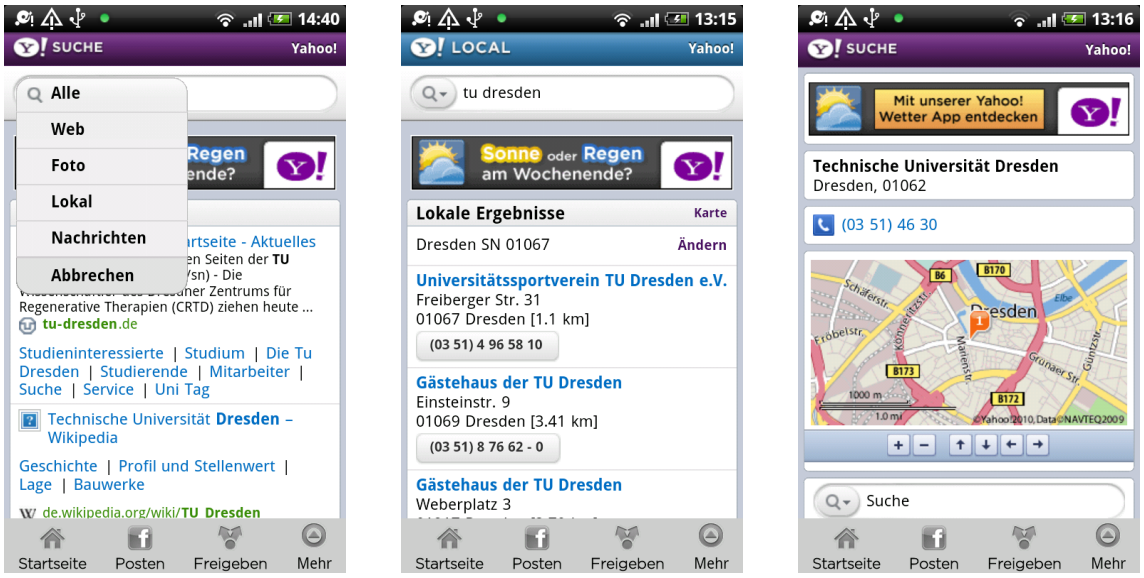
Code 3.11: Microsoft Bing: Location Ressource

Locations API ist dabei der Einstiegspunkt für Entwickler[6], welche ortsbezogene Daten anfordern wollen. Es bestehen auch hier drei verschiedene Möglichkeiten auf Orte zuzugreifen: über eine eingegebene Adresse, über die Koordinaten und über die Eingabe eines Suchterms. Ergebnisse werden als „Resources“ zurück geliefert und enthalten neben genauen Adress- und Koordinatendaten auch ein „confidence“-Level, eine Boundingbox und einen Datentyp (siehe Codebeispiel 3.11). Über den „confidence“ Wert kann der Entwickler besser und vor allem selber über die Wichtigkeit eines Suchergebnisses entscheiden und es dementsprechend hervorheben oder anders anordnen. Die Boundingbox ist eine weitere Erleichterung bei der Arbeit mit Koordinaten. Sie gibt die maximale Ausdehnung eines Objektes an. Boundingboxen dienen zur Erkennung von Kollisionen oder fungieren als Entscheidungskriterium über die Sichtbarkeit im aktuellen Kartenausschnitt. Der „EntityType“ dient zur Klassifizierung des Ergebnisses. Dabei ist die Menge aller Typen auf eine festgelegte Liste mit 192 Einträgen begrenzt.

[6] `http://msdn.microsoft.com/en-us/library/ff701715.aspx` (Zugriff am 29.10.2011)

3.4 YAHOO! MAPS

Der dritte Anbieter von Karten und damit verknüpften Informationen ist Yahoo!. LBS werden über die sogenannte „Lokal Suche“[7] für den Desktopnutzer zur Verfügung gestellt. Die offensichtlichen Funktionen unterscheiden sich dabei nicht von den bisher vorgestellten. Es gibt zusätzliche Einschränkungsmöglichkeiten und Filter für die Suche, welche die Übersichtlichkeit verbessern sollen. Zusätzlich existiert auch für Yahoo! eine eigenständige Android App, die nachfolgend kurz beschrieben wird.



(a) Auswahlmöglichkeiten / Filter für die Suche

(b) Suchergebnisse

(c) Detailansicht mit Kartenausschnitt

Abbildung 3.5: drei Screenshots der Yahoo! Android App

3.4.1 Android App

Die analysierte App liegt zum aktuellen Zeitpunkt in der Version 0.9.4.1 vor. Es handelt sich bei dieser App ebenfalls um eine Anwendung, die alle Funktionen der Yahoo! Suche vereint. Die „Lokal Suche“ (Auswahlmöglichkeit in Abbildung 3.5a) ist dabei nur ein Teil der kompletten Anwendung und kann als Filter auf die Suchergebnisse angewandt werden. Ein Nachteil der App ist, dass Kartenausschnitte nur als statische Kacheln angezeigt werden und wenig interaktiv zu bedienen sind (siehe Screenshot 3.5c). Es wird überdurchschnittlich viel Wert auf die Verknüpfung mit sozialen Netzwerken gelegt, da man in jeder Ansicht die Möglichkeit hat, den aktuellen Inhalt zu „teilen“ oder zu „posten“. Ergebnisse der Suche werden wie von anderen Apps gewohnt, mit Adresse und wenigen Details angezeigt (Abbildung 3.5b). Sofern eine Telefonnummer vorhanden ist, kann diese per Button gewählt werden.

```
<?xml version="1.0" encoding="UTF-8"?>
```

[7] `http://de.local.yahoo.com/` (Zugriff am 30.10.2011)

```
<ResultSet version="1.0">
  <Error>0</Error>
  <ErrorMessage>No error</ErrorMessage>
  <Locale>us_US</Locale>
  <Quality>87</Quality>
  <Found>1</Found>
  <Result>
    <quality>87</quality>
    <latitude>37.416275</latitude>
    <longitude>-122.025092</longitude>
    <offsetlat>37.416397</offsetlat>
    <offsetlon>-122.025055</offsetlon>
    <radius>500</radius>
    ...weitere Attribute...
  </Result>
</ResultSet>
```

Code 3.12: Yahoo! Placefinder: Ergebnisliste mit einem Eintrag

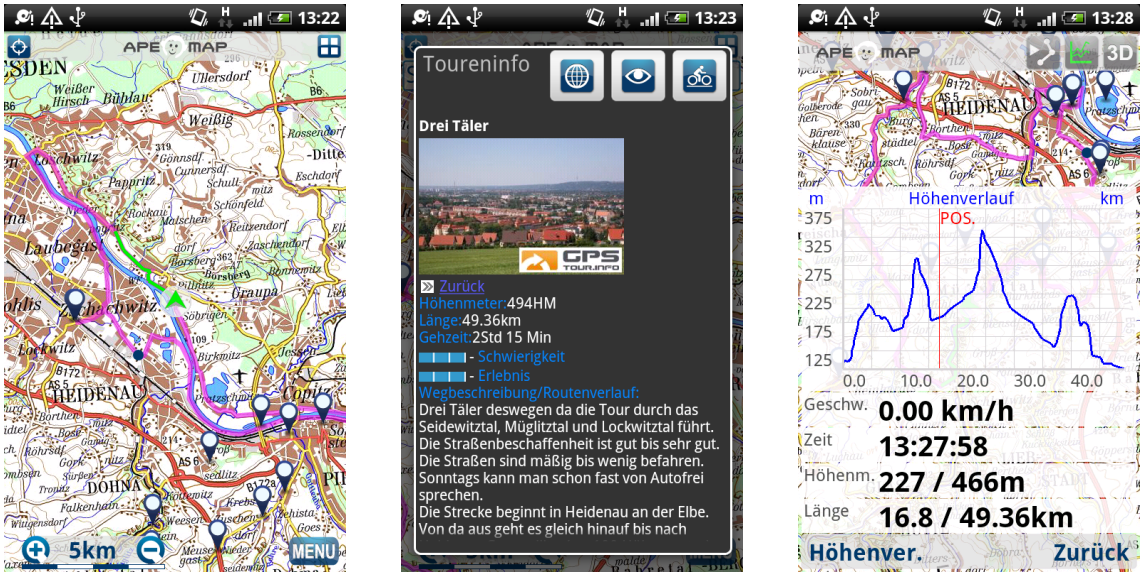
3.4.2 Developer API

Die API, welche die wichtigsten ortsbezogenen Funktionen für einen Entwickler bereitstellt, wird **Yahoo! PlaceFinder** genannt und bietet im Vergleich zu den API's der Abschnitte 3.2.3 und 3.3.2 einen weiteren beachtenswerten Rückgabewert[8].
Über eine umfangreiche Auswahl von „location parameter" ist es möglich, sowohl Suchanfragen, als auch Adress- und Koordinatensuchen durchzuführen. Die Angabe der Parameter entscheidet dabei, wie die Anfrage behandelt wird und welche Ergebnisse geliefert werden. Interessant dabei ist die Angabe eines „Quality"-Wertes, der angibt, welche Suchinformationen zu dem jeweiligen Ergebnis geführt haben. Im Codebeispiel 3.12 bedeutet der zurückgegebene Wert 87, dass die Adressinformation der Suchanfrage zu diesem Ergebnis geführt hat.

3.5 APE@MAP

ape@map ist eine von Onyx Technologie OG hergestellte Outdoor-Navigations-Software [EJ10]. Sie wurde als Nischenprodukt für die Navigation speziell für Wanderer, Skifahrer und Mountainbiker entwickelt. Die Anwendung wird hinsichtlich ihrer Flexibilität und besonderen Funktionalität in diesem Kapitel analysiert.

[8] `http://developer.yahoo.com/geo/placefinder/guide/requests.html` (Zugriff am 30.10.2011)



(a) Vorschlag für eine Wanderroute von ape@map

(b) Detailanzeige eines Markers

(c) Höhenprofil der Wanderroute

Abbildung 3.6: drei Screenshots der ape@map Android App

3.5.1 Konzeption

Im Vorfeld wurden marktwirtschaftliche Potentiale einer solchen Navigations-Software genaustens untersucht. Die **Zielgruppen** wurden auf die drei Urlauberkreise: Wanderer, Mountainbiker und Skitourengeher eingegrenzt, da das Hauptvermarktungsgebiet die österreichischen Alpen sind. Die Zahlen der inländischen und ausländischen Touristen waren für diese Analyse ausschlaggebend.
Des Weiteren sollte eine kostengünstige Software entwickelt werden, die ohne teure Spezialgeräte funktioniert. Um einen möglichst großen Anteil der aktuell eingesetzten Systeme zu bedienen, wurde der **Marktanteil der aktuell bestehenden Handy-Betriebssysteme** bewertet. Ergebnis dieser Analyse war die anfängliche Implementierung für Symbian, RIM BlackBerry und Android. Später wurde die erfolgreiche Software für Nokia, Sony Ericsson Handys und das iPhone portiert. Windows Mobile 7 ist ebenfalls geplant[9].
Von Beginn an wurde bei der Entwicklung darauf geachtet, dass die eingetragenen Routen mit verschiedenen Möglichkeiten der Gamification **erweiterbar** sind. Das heißt, dass Wanderrouten mit Spielen wie „Schnitzeljagd“ oder „Geocaching“ für die Touristen anreizvoller gestaltet werden können. Zusätzlich besteht die Option, die Software mit Extrafunktionen für andere Urlaubsregionen auszustatten.
Als besondere Eigenschaften einer Route wird dem Nutzer Folgendes angezeigt:

das Höhenprofil sowohl als Diagrammansicht (Abbildung 3.6c) als auch in der 3D-Karte,

die Schwierigkeit der Strecke als Bewertungshinweis (Abbildung 3.6b) und

die Eignung für die drei verschiedenen Nutzergruppen (Abbildung 3.6b).

[9] `http://www.apemap.de/` (Zugriff am 03.11.2011)

3.6 OPENSTREETMAP

OpenStreetMap ist ein Open Source-Projekt, welches frei nutzbare Geodaten für alle Nutzer zur Verfügung stellt[10]. Durch die Zusammenarbeit vieler Unterstützer kann der Datenbestand ständig verbessert und aktuell gehalten werden. Zu den gepflegten Daten gehören nicht nur Straßen und Orte sondern auch alle für LBS relevanten, ortsbezogenen Informationen. Die Daten des OpenStreetMap-Projektes werden in Postgre-Datenbanken gespeichert. Den Zugriff und damit auch die Darstellung der Karten erfolgt mit Hilfe vieler verschiedener Softwareprojekte. Die eigentliche API ist aber über das OpenStreetMap-Projekt definiert. Durch den Open Source-Charakter des Projektes ist die Analyse der API hinsichtlich dem Einfügen von Daten besonders interessant.

3.6.1 Developer API

Die aktuelle Version der API ist v0.6[11]. Die API zeichnet sich durch die Möglichkeit aus, alle Informationen und Elemente einer angegebenen Boundingbox zu erhalten. Das Konzept von „relations“ ermöglicht das Wiedergeben aller Wegpunkte oder anderweitig verknüpften Elemente. Eine einfache Suche nach selbst definierten Begriffen wird durch die API erschwert. Das System ist mit Hilfe von „changesets“ stärker auf die Pflege (create, change, delete) der Daten ausgelegt. Dadurch wird jede Änderung am Datenbestand mit einem Kommentar und dem Ersteller markiert, ähnlich eines Versionierungssystems.

3.7 COMMUNITIES

LBS sind nicht nur bei den vorgestellten Kartenanbietern und Anwendungen immer populärer geworden. Auch und vor allem die unterschiedlichsten Communities versuchen mit ihren Angeboten möglichst viele Nutzer zu binden. Das Prinzip ist dabei immer sehr ähnlich: Der Nutzer meldet sich mit seinem aktuellen Standort bei der jeweiligen App an und bekommt im Gegenzug Vergünstigungen oder Sonderangebote von Händlern in der unmittelbaren Umgebung.

[10] `http://wiki.openstreetmap.org/wiki/DE:Main_Page` (Zugriff am 03.11.2011)
[11] `http://wiki.openstreetmap.org/wiki/API_v0.6` (Zugriff am 03.11.2011)

Dienst / Service	Qype[1]	foursquare[2]	Groupon[3]	Gowalla[4]	Yelp[5]	Facebook Places[6]
Suche	x	x	x	x	x	x
erweiterte Suche	x		x		x	
Checkin	x	x		x	x	x
Posten auf anderen Plattformen	x	x				x
Freunde	x	x		x	x	x
Bewertung	x	x			x	x
Kommentar	x	x	x	x	x	x
Deals	x	x	x		x	
Bilder	x	x	x	x	x	x
Belohnungen	x	x		x		

[1] `http://apidocs.qype.com/` (Zugriff am 03.11.2011)
[2] `https://developer.foursquare.com/docs/` (Zugriff am 03.11.2011)
[3] `http://sites.google.com/site/grouponapiv2/home` (Zugriff am 03.11.2011)
[4] `http://gowalla.com/api/docs` (Zugriff am 03.11.2011)
[5] `http://www.yelp.com/developers/documentation` (Zugriff am 03.11.2011)
[6] `http://developers.facebook.com/docs/reference/api/` (Zugriff am 03.11.2011)

Tabelle 3.1: Vergleich der populärsten LBS-Communities

In der Tabelle 3.1 sind die wichtigsten Communities und ihre jeweils unterstützten Dienste aufgelistet. Erklären lassen sich diese Dienste wie folgt:

Die Suche meint die Möglichkeit durch Angabe von Koordinaten und einem Radius oder einem Ort eine Suchanfrage versenden zu können, um Ergebnisse aus der Umgebung zu erhalten.

Die erweiterte Suche hingegen ermöglicht das Suchen mit weiteren Parametern, wie zum Beispiel Kategorien, aber auch maximale Anzahl und Sortierung der Ergebnisse.

Der Checkin ist die Funktion, sich an einem Ort anzumelden.

Das Posten auf anderen Plattformen erlaubt es dem Nutzer, seinen Standort direkt auf andere soziale Netze zu übertragen.

Durch Freunde bzw. Kontakte ist die Möglichkeit gegeben, sich mit ihnen in der jeweiligen Community selbst zu verknüpfen.

Die Bewertung wird immer für den jeweiligen Ort generiert.

Ein Kommentar zum Ort kann der Nutzer abgeben, wenn er sich am Standort angemeldet hat.

Deals sind Vergünstigungen oder Sonderangebote, die dem Nutzer angeboten werden.

Bilder helfen ähnlich wie ein Kommentar und die Bewertung den Standort zu beschreiben.

Belohnungen werden dem Nutzer für bestimmte Aktionen, Anzahl von Checkins verliehen und erweitern die Community mit einem Spiel- und Level-Up-Charakter.

Es stellt sich heraus, dass alle untersuchten Communities neben der notwendigen Suche auch Bilder und Kommentare der Nutzer zum Beschreiben der Orte und Angebote verwenden. Aber auch die anderen Dienste werden von fast allen angeboten, so dass nur wenige kleine Unterschiede im Gesamtbild entstehen.

3.8 KRITIKEN

Die bisherigen Lösungen zeigen umfangreiche Möglichkeiten für den Einsatz von Location-Based Services. Im Outdoor-Bereich gibt es eine Vielzahl von Funktionen, die dem Nutzer die gegebenen Geodaten bestmöglich aufbereiten. Die Plattformen und Softwarelösungen ähneln sich größten Teils. Gemeinsamkeiten sind die Suchfunktionen für die aktuelle Position des Nutzers, das Erkunden ausgewählter Umgebungen und Zusatzfunktionen wie Navigation sowie Checkins bei bekannten Orten. Signifikante Unterschiede zwischen Desktopimplementierungen und Anwendungen für mobile Geräte existieren kaum.

Das Problem der bestehenden Anwendungen ist, dass sie keine Integration für Indoor-LBS bieten. Es existieren weder Suchfunktionen noch andere weiter gehende Funktionalitäten. Lediglich Google Maps stellt bereits Kartenmaterial für Innenräume zur Verfügung. Dieses ist nicht für Softwareentwickler verfügbar, obgleich die verfügbaren Outdoor-API's sehr gut strukturiert sind. Die verwendeten Frameworks sind nicht um die nötigen Funktionen erweiterbar um die Aufgabenstellung dieser Arbeit zu erfüllen. Auch die Open Source-Ansätze von OpenStreetMap, sind von der Datenstruktur her noch nicht auf Indoor-Services vorbereitet.

Der MapBiquitous-Prototyp aus Kapitel 2.4 bildet die besten Grundlagen für das angestrebte integrierte Framework für Indoor und Outdoor-LBS. Die genauen Anforderungen der Umsetzung werden im Kapitel 4 analysiert

4 ANFORDERUNGEN

Ziel der Arbeit ist die Entwicklung eines Frameworks für Indoor und Outdoor-LBS. Das konzipierte Framework soll für das MapBiquitous-Projekt implementiert werden, um das bereits vorhandene System um die Funktionalitäten von LBS im Indoor-Bereich zu erweitern. Die vorhandenen Gebäudeinformationen werden dazu um weitere Datenstrukturen ergänzt, um dem Nutzer eine vielfältigere und komfortablere Nutzung der Informationen ermöglichen zu können. Es kommt dabei darauf an, dass mit Hilfe des Frameworks Funktionalitäten implementiert werden können, die durch Freigabe des aktuellen Standorts die sinnvollsten ortsbezogenen Outdoor-Dienste für den Indoor-Bereich ermöglichen.

4.1 TECHNISCHE UMSETZUNG, ARCHITEKTUR

Dieses Kapitel zeigt die Anforderungen im Bereich der Architektur und Datenverwaltung. Es werden die analysierten Probleme aufgezeigt und entsprechende Anforderungen formuliert, um alle notwendigen Daten erfassen zu können.

Datenstruktur Der wichtigste Punkt und Grundvoraussetzung für die erweiterte Einbindung von LBS ist die Anpassung der aktuellen Architektur. Dazu müssen die gewonnenen Erkenntnisse zur erforderlichen Datenstruktur umgesetzt werden. Das heißt, die Haltung der Daten auf den Gebäudeservern muss überprüft und angepasst werden. Da die aktuelle Spezifikation auf den Standards von OpenGIS beruht und sich dieses Web Service Framework als beste Plattformgrundlage durchgesetzt hat, soll auch das konzipierte Framework zu diesen Standards konform sein. Die aktuelle Datenstruktur eines Informationspunktes geht aus dem in [Sch09a] vorgestellten Schema hervor und muss entsprechend angepasst werden. Das Einpflegen von Testdaten bzw. das Migrieren der vorhandenen Daten ist Grundvoraussetzung, um das System anschließend evaluieren zu können und den Datenstand zu erhalten.

Plattformunabhängigkeit Auf Grund der Umfangreichen Nutzungsmöglichkeiten des zu entwickelnden Frameworks soll es plattformunabhängig konzipiert werden. Das hat den Vorteil, dass für die Vielzahl der existierenden mobilen Endgeräte eine entsprechende Implementierung der MapBiquitous-App angestrebt werden kann. Dieser Umstand räumt auch die Möglichkeit weiterer zusätzlicher Funktionen und Erweiterungen ein, ohne, dass eine Entwicklungsplattform ausgeschlossen wird. Anhand der sich rasant weiter entwickelnden mobilen Betriebssysteme ist dies die einzige Möglichkeit, auch in Zukunft auf dem Framework aufzubauen und Software für den Indoor-Bereich mit LBS auszustatten.

Implementierung Um die Funktionalität des Frameworks zu testen, ist eine beispielhafte Implementierung unabdingbar. Sie soll auf dem Androidbetriebssystem umgesetzt werden. Die Evaluierung kann somit auf den originalen Daten der aktuellen MapBiquitous-App angestrebt werden.

Kernfunktionalität Das entstandene Framework verwirklicht verschiedene neue Funktionen, die teilweise in ähnlicher Form bereits für den Outdoor-Bereich bestehen. Sie müssen konform zu den aktuellen Nutzergewohnheiten sein, um einen Bruch der Usability zu vermeiden. Hierzu zählt, dass dem Nutzer aber auch dem Entwickler die in Abschnitt 2.2.2 und Kapitel 3 vorgestellten Möglichkeiten zur Verfügung gestellt werden sollen. Für den Anwender sind dies die Suchfunktionen nach geeigneten Diensten und Informationen abhängig von seinem Standort. Es zählen ebenfalls die erläuterten mobilen Services, wie Navigation, dazu. Für den Softwareentwickler werden Möglichkeiten zur unkomplizierten Erweiterung des Frameworkes bereitgestellt. Schnittstellen für Datenabfragen stehen zur Verfügung und sind gut dokumentiert. Dabei ist darauf zu achten, dass es sich um ein Framework handelt, bei dem der Entwickler die Option hat, die Funktionen zu nutzen, sie aber nicht zwingend in seiner Software einbinden muss. Auf Grund der überschaubaren Projektgröße und der eher geringen Zahl an Entwicklern, wird eine Aufteilung der Kernfunktionen auf verschiedene API's, wie es bei Googles „3.2.3 Developer API" praktiziert wurde, nicht notwendig sein.

Interaktionsmöglichkeiten Um die bereits beschriebene Plattformunabhängigkeit zu gewährleisten und einen einheitlichen Zugriff auf das Framework zu ermöglichen, muss der Zugriff auf die Funktionalitäten über wohl definierte Schnittstellen geschehen. Dem Entwickler werden die entsprechenden Methoden zur Verfügung gestellt, so dass er sich selbst nicht um die Anbindung der Services kümmern muss. Die initiale Konfiguration des Frameworks sollte durch ausgelagerte Settings realisiert werden. Das erleichtert das Anbinden anderer Services und fügt wichtige Grundeinstellungen in einer zentralen Datei zusammen.

4.1.1 Zusammenfassung

A1.1 die Konzeption sollte plattformunabhängig sein

A1.2 Entwicklungen eines Prototypen für Androidplattformen

A1.3 Gebäudeserver einrichten und konfigurieren

A1.4 Datenhaltung für LBS anpassen

A1.5 Datenstruktur eines Informationspunktes anpassen - Erweiterung nach [Sch09a]

A1.6 Kernfunktionalitäten implementieren

A1.7 UI als Teil des Frameworks integrieren

A1.8 Settings unabhängig und separat konfigurierbar auslagern

4.2 SCHNITTSTELLE ZUM SERVER, DIENSTDEFINITION

Das Kapitel umfasst alle Anforderungen, die an Schnittstellen des Servers gestellt werden. Sie werden abgeleitet aus den Funktionen, die zum Abfragen der Dienste benötigt werden. Mit Hilfe der definierten Anforderungen soll eine RESTful API für die Gebäudeserver entstehen.

Schnittstellenerweiterung der Gebäudeserver Die MapBiquitous-Gebäudeserver besitzen in ihrer aktuellen Konfiguration bereits Schnittstellen, um grundlegende Funktionen eines WFS zur Verfügung zu stellen. Diese Konfiguration muss überarbeitet und ergänzt werden. Ziel ist es, dass die verschiedenen ortsbezogenen Dienste die korrekte Schnittstelle des Servers ansprechen und die aufgearbeiteten Ergebnisdaten im XML-Format zur Verfügung gestellt bekommen.

Positionsbestimmung Für den Ansatz die Positionsbestimmung des Clients mit der Fingerprinting-Methode durchzuführen, ist eine Schnittstelle notwendig. Sie nimmt die Parameter des Verfahrens entgegen und gibt als Ergebnis die Positionsberechnung zurück. Der Client kann diese Position für die Darstellung oder für weitere Anfragen verwenden. Um diesen Ansatz zur Ortung des Clients zu verbessern, wird eine Schnittstelle benötigt, die sowohl Fingerprinting-Parameter als auch die Position entgegen nimmt. Das Interface zur Kommunikation mit einem Server sollte so generisch wie möglich angelegt werden, um anderen Verfahren, die einen externen Server zur Positionsbestimmung heranziehen, den Datenaustausch zu ermöglichen. Sofern der Client seine aktuelle Position durch ein anderweitiges, client-seitiges Ortungsverfahren erhält, ändert sich nichts an den definierten Schnittstellen.

Suche und Abfrage von Diensten Auf dieser Grundlage entstehen die weiteren Anforderungen. Funktionen zum Auffinden von Diensten in der näheren Umgebung, definiert durch einen Radius, müssen genauso bereitgestellt werden, wie das Suchen an einem entfernten Ort. Dabei entstehen jeweils zwei Möglichkeiten: Das direkte Suchen mit und ohne einem eingegebenen Suchterm. Das Suchen ohne Stichwörter entspricht dabei der Auflistung aller Services im festgelegten Radius. Ungeachtet des Radiuses sollten „globale" Informationen zum gewählten Gebäude jederzeit abfragbar sein. Dazu gehören zum Beispiel der Name und die Adresse des Gebäudes.

Datenerfassung Die Analyse hat gezeigt, dass durch Crowdsourcing die Datenaktualität wesentlich verbessert wird und detailgetreue Informationen zur Verfügung stehen. Dieses Potential muss genutzt werden, in dem über eine Schnittstelle auch Datensätze hinzugefügt und bearbeitet werden können. Diese Schnittstelle verdient besondere Sicherungsmaßnahmen, um den Datenmißbrauch zu verhindern.

4.2.1 Zusammenfassung

A2.1 Schnittstellenerweiterung nach [Sch09a]

A2.2 Schnittstellen zum Datenaustausch mit externen Anbietern, welche die Positionsbestimmung (z.B. durch Fingerprinting) ermöglichen

A2.3 Schnittstelle um Daten per Crowdsourcing einzufügen

A2.4 Auffinden einer Einrichtung (POI)

A2.5 Suchen mit und nach Mustern

A2.6 Was ist in der Nähe von mir oder einem definierten Ort

A2.7 Informationen zum aktuellen Gebiet oder Gebäude

A2.8 Eingabemöglichkeiten zum Navigieren (von Punkt zu Punkt, zum POI)

A2.9 Eingabe und Erstellen von Daten

4.3 UI DES CLIENTS, DARSTELLUNG

Nicht zuletzt müssen alle getroffenen Anforderungen an den Server und die Datenstruktur auf dem mobilen Endgerät widergespiegelt werden können. Dazu muss das Framework einfache Methoden bereitstellen, die die erhaltenen Daten für die Darstellung aufbereiten. Andererseits muss die Eingabe der Client-Position und der Suchterme erfolgen und noch auf dem Client auf die Übermittlung zum Server vorbereitet werden. Ein wichtiges Element wird die Darstellung der Karte sein. Sie muss intuitiv bedienbar sein und nach einer erfolgreichen Suche die Ergebnisse enthalten. Alle Gebäudeinformationen, die ohne erfolgte Suche dem Nutzer präsentiert werden, müssen strukturiert ohne Verzögerung in die Karte eingebettet werden, um eine User Experience zu erhalten. Zusätzlich zur Kartendarstellung ist eine Detailansicht und eine Listenansicht der gefundenen Dienste sinnvoll. Um den Nutzer gewisse Personalisierungsmöglichkeiten zu erlauben, ist ein Menü mit Grundeinstellungen notwendig. Es soll aber nur eine begrenzte Anzahl an Konfigurationen bereit gestellt werden, um die Anwendung nicht unübersichtlich erscheinen zu lassen. Als Eingabemöglichkeiten sollten die Grundfunktionen des Mobilgerätes ausgeschöpft werden. Dabei gilt es die Funktionalität im Framework so zu gestalten, dass Interaktionen auf verschiedenen Plattformen ähnlich genutzt werden können.

4.3.1 Zusammenfassung

A3.1 grafische Umsetzung der Schnittstellen durch Eingabemöglichkeiten

A3.2 Darstellung der Karte

A3.3 Darstellung der Suchergebnisse in der Karte

A3.4 Darstellung der Suchergebnisse als Liste

A3.5 Darstellung eines Suchergebnisses als Detailansicht

A3.6 UI nutzerfreundlich und übersichtlich gestalten

A3.7 UX beachten

A3.8 einfache Konfigurationsmöglichkeiten und Personalisierungsoptionen

4.4 ZIELGRUPPE

Da es sich in erster Linie um die Entwicklung eines Frameworks handelt, muss auf die sorgfältige Dokumentation geachtet werden. Funktionen müssen nach gängigen DocStyle-Regeln beschrieben werden. Um die Nutzbarkeit zu Evaluieren, ist eine Analyse mit Testpersonen in der Rolle von Entwicklern sinnvoll. Eine weitere Zielgruppe stellen die zukünftigen Nutzer des angewandten Frameworks dar. Hier sind Usability-Tests notwendig um die Nutzbarkeit der entwickelten Software zu analysieren.

4.4.1 Zusammenfassung

A4.1 Dokumentation für Entwickler

A4.2 Usability-Tests für Nutzer

4.5 ZUSAMMENFASSUNG

Abschließend lassen sich die Anforderungen und die damit notwendig werdenden Änderungen wie folgt zusammenfassen:

Aus architektonischer Sicht sollte ein Framework entstehen, welches die bestehenden Gebäudedaten verwalten und verarbeiten kann. Das Entwickeln neuer Komponenten zur Anbindung weiterer Dienste sollte so einfach wie möglich gestaltet werden. Gebäudeserver und clientseitige Modells müssen für die neue Struktur vorbereitet und konfiguriert werden. Dabei werden Konstanten an zentraler Stelle ausgelagert um eine flexible Anpassung zu ermöglichen.

Das Zusammenspiel zwischen dem Client und dem Server wird über Schnittstellen definiert und gesteuert. Sie sollten die diskutierten Abfragemöglichkeiten realisieren und ebenfalls zur Dateneingabe geeignet sein. Für jede der besprochenen Suchen wird eine separate Schnittstelle erstellt, wobei doppelte Implementierungen zu vermeiden sind.

Da die Nutzeroberfläche die meisten der neuen Funktionen visualisieren muss, ergeben sich auch in diesem Bereich umfassende Anforderungen. Schnittstellen müssen bedient und teilweise durch Nutzereingaben befüllt werden. Für alle neuen Informationen, welche dem Nutzer als Ausgaben präsentiert werden, werden Konzepte für die Darstellung benötigt. Dabei sollten Usability- und Design-Standards beachtet und angewandt werden.

Am Ende der Konzeption und Entwicklung steht eine Evaluation um die Ergebnisse der Arbeit zu prüfen.

5 KONZEPTION DES FRAMEWORKS

Das folgende Kapitel beschreibt das Konzept bezüglich der umzusetzenden Anforderungen (Kapitel 4). Dabei wird die theoretische Vorgehensweise für die Entwicklung des Frameworks festgelegt. Ziel soll es sein, mit Hilfe der vorangegangenen Untersuchungen ein Framework für integrierte Indoor/Outdoor Location-Based Services zu entwickeln. Das Konzept, welches als Ergebnis dieses Kapitels entsteht, ist Grundlage für die weitere Entwicklung des MapBiquitous-Prototypen. Es soll unter 6. „Implementierung“ umgesetzt und damit überprüft werden.

Um die Konzeption möglichst umfassend zu gestalten, werden drei Bereiche gesondert betrachtet und jeweils spezielle Lösungen entwickelt. Der erste Bereich (Abschnitt 5.1) legt die nötigen Änderungen an der Datenstruktur von Datenbank und Client fest. Nachfolgend werden in Kapitel 5.2 die Package-Architektur und Schnittstellen des Frameworks definiert. Zum Abschluss wird die neue Nutzeroberfläche durch entsprechende Interaktionsmuster und Untersuchungen in Abschnitt 5.3 detailliert beschrieben.

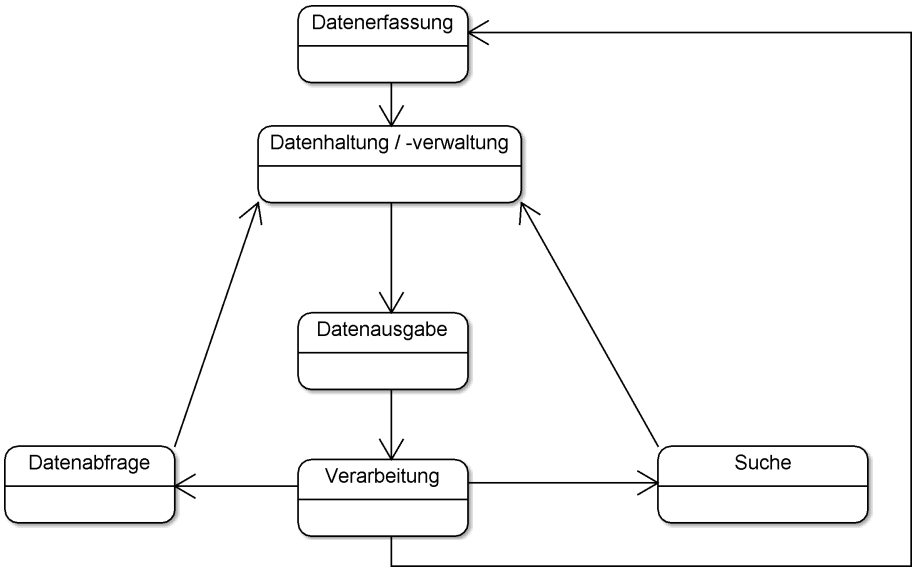


Abbildung 5.1: allgemeine Abläufe in MapBiquitous

Einen Gesamtüberblick über das System zeigt die Grafik 5.1. Als zentraler Bestandteil ist die Datenhaltung bzw. Datenverwaltung anzusehen. Informationen können über die Datenerfassung eingefügt werden und stehen ab diesem Zeitpunkt allen anderen Diensten zur Verfügung. Durch eine einfache Abfrage können gezielt Geodaten aus der Datenbank gewonnen werden. Sie werden ausgeliefert und durchlaufen die Verarbeitung - zum Beispiel bei der Darstellung. Eine andere Möglichkeit besteht, indem der Datenbestand durchsucht wird. Ergebnisse der Suche werden vom Gebäudeserver zurück gegeben und stehen zur Verarbeitung bereit. Während und nach der Verarbeitung kann es notwendig werden, weitere Daten anzufordern. Das ist der Fall, wenn nach einer Suche Details geladen werden sollen oder sich der dargestellte Kartenausschnitt ändert und weitere Gebäudeteile abgerufen werden müssen. Es entsteht eine Art Kreislauf. Auch Daten, die abgefragt und aktualisiert wurden, können über die Datenerfassung dem System wieder zugeführt werden. Sie stehen ab diesem Zeitpunkt dem System in aktualisierter Version wieder zur Verfügung.

5.1 DATENSTRUKTUR

Die Datenstruktur des bisherigen Projektes ist ausschließlich auf das Laden und Darstellen von Etagen und Gebäudegrundrissen ausgelegt. Serverseitig werden die Geodaten in einer PostGIS-Datenbank verwaltet und über den Gebäudeserver an den Client ausgeliefert. Jedes Gebäude verfügt dabei prinzipiell über eine eigene Url und damit über einen eigenen Gebäudeserver. Der Client erhält die angefragten Daten und stellt sie für den Nutzer dar. Die Anforderungsanalyse (Kapitel 4) hat ergeben, dass die Umsetzung weiterer Funktionen client- und serverseitig unabdingbar ist, um ein flexibles Framework für Location-Based Services umzusetzen. Dementsprechend muss auch die Struktur der Datenhaltung angepasst werden (Anforderung A1.4 und A1.5). Im folgenden Kapitel wird auf die notwendigen Änderungen eingegangen.

5.1.1 Erweiterung der Datenbank

Die vorhandene PostGIS-Datenbank stellt weiterhin alle Geodaten zur Verfügung. Sie bietet die beste und aktuellste Möglichkeit, die anfallenden Daten serverseitig zu verwalten. Der Zugriff auf die Datenbank wird über den entsprechenden Gebäudeserver konfiguriert (siehe Kapitel 5.2.2). Es sind daher keine direkten Schnittstellen zwischen dem Client und der Datenbank notwendig. Dieses Vorgehen hat den Vorteil, dass jederzeit die Datengrundlage ausgetauscht werden kann. Es ist ebenfalls mit einfachen Mitteln möglich, einen Datenbank-Dump zu erstellen. Dieses Abbild kann als Sicherung oder als Datengrundlage auf anderen Servern eingespielt werden. Für den Fall, dass die Datenbanktechnologien weiter entwickelt werden oder aus Lizenzgründen andere Datenspeichermechanismen eingesetzt werden müssen, kann dies unabhängig von der restlichen Architektur geschehen.

Die aktuelle Datenbank enthält bisher Testdaten für vier Gebäude des Universitätscampus in Dresden. Sie umfassen Geodaten für alle Etagen und Räume der Gebäude. Die somit vorhandenen Koordinaten und damit verbundenen Metadaten sind für die Evaluation ausreichend. Es

fehlen aber noch Daten zu POI's außerhalb von Gebäuden. Da die Suche auch solche Daten berücksichtigen soll, sind entsprechende Testdaten notwendig. Dazu muss der Datenbestand um das Layer der Points of Interest erweitert werden. Es können Beispieldaten wie etwa Papierkörbe, Fahrradständer oder Sitzgelegenheiten auf dem Campus aufgenommen werden. Da die Einbindung externer LBS, wie zum Beispiel der Daten aus OpenStreetMap, geplant ist, kann von Anfang an darauf geachtet werden, dass sich die Beispieldaten nicht doppeln sondern gegenseitig ergänzen.

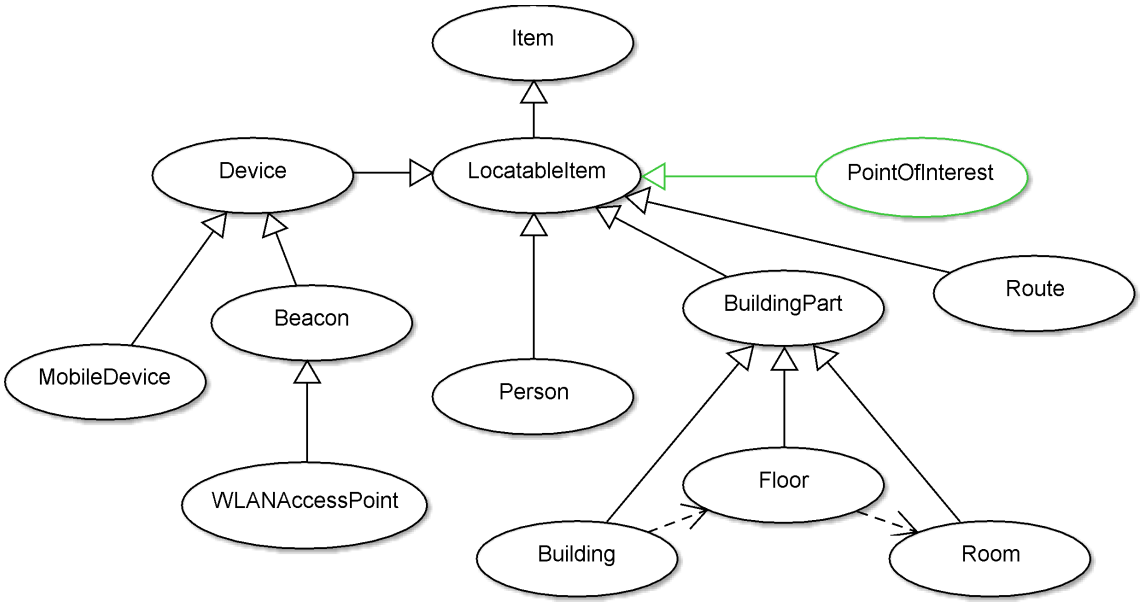


Abbildung 5.2: Client side Modells (Ausschnitt)

5.1.2 Client-seitige Erweiterungen

Das im vorherigen Abschnitt (5.1.1) beschriebene Datenmodell spiegelt sich auch auf der Client-Seite wieder. In Kapitel 2.4 wurde die aktuelle Struktur an Hand der Abbildung 2.5 beschrieben. Dieses Modell wird wie in Abbildung 5.2 erweitert. *PointOfInterest* (POI) ist dabei die neu eingefügte Unterklasse des *LocatableItems*. Die Architektur ermöglicht es, auf dem Client eine einheitliche Darstellung von interessanten Punkten zu generieren. So lässt sich auch ein Raum, welcher einen besonderen Service anbietet (beispielsweise ein Kopierraum), zu einem POI transformieren. Alle Ergebnisse einer Suche oder Abfrage können so dem Nutzer konsistent präsentiert und dargestellt werden.

5.2 SCHNITTSTELLEN / ARCHITEKTUR

Das folgende Kapitel gibt Aufschluss darüber, wie die Architektur des MapBiquitous-Clients angepasst wird, um Entwurfskonzepte mit hoher Flexibilität und Funktionalität zu vereinbaren. Die Architektur soll nach Anforderung A1.1 auf andere Plattformen übertragbar sein. Dazu ist es notwendig, allgemeingültige Entwurfsmuster in der Konzeption und Programmierung zu verwenden.

Eine weitere Forderung, die aus dem „Framework-Gedanken“ hervorgeht, ist die Erweiterbarkeit und Wiederverwendbarkeit des Projektes. In den nachfolgenden Abschnitten wird besonderer Wert darauf gelegt, die Forderungen zu erfüllen und die empfohlenen Entwurfsmuster zu verwenden.

5.2.1 Client-Architektur

In Kapitel 2.4 wurde deutlich gemacht, dass die bisherigen Komponenten des Clients (Loader, Renderer und Locator) überwiegend lose und „ungeordnet“ miteinander kommunizieren und Daten austauschen. Dieses Vorgehen erschwert die Pflege des Gesamtsystems. Auch die Erweiterbarkeit ist eingeschränkt, da an vielen Stellen Änderungen vorgenommen werden müssen, um weitere Funktionalität oder Komponenten hinzuzufügen. Ziel der Client-Architektur ist eine weitgehende Unterteilung in Packages mit festgelegten Aufgabenbereichen. Mit einem Package ist der herkömmliche Mechanismus gemeint, Javaklassen in Namensräume (engl. namespaces) einzuteilen. Die strikte Organisation der einzelnen Programmkomponenten ist auch dahingehend wichtig, da bei einem integrierten Indoor- und Outdoor-LBS externe Services als Datenlieferanten hinzukommen.

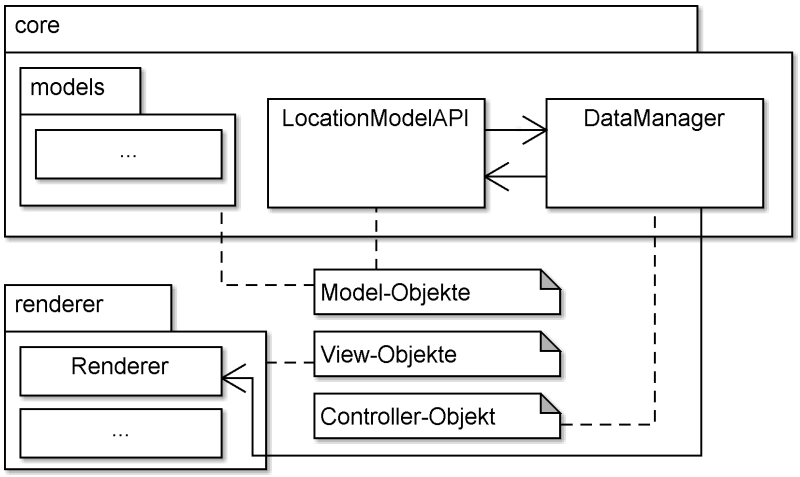


Abbildung 5.3: Darstellung des MVC-Pattern als Teil der Architektur

Das MVC-Pattern (Model View Controller-Pattern) existiert seit Langem in der Softwareentwicklung und fördert die eben beschriebene Trennung nach bestimmten Funktionsbereichen. Dieses Entwicklungsmuster schreibt in erster Linie die Trennung der drei Objekte, die seinen Namen bilden, vor und schafft so die Grundlage für viele Web- und Application-Frameworks. Die sogenannten Objekte sind nicht als einzelne Klassen zu verstehen, vielmehr gibt es mehrere Klassen, die die Eigenschaften des Model-Objekts bzw. des View- und Controller-Objektes erfüllen (nach [GHVJ09] und [Bur97]). Es geht hierbei nur um die funktionelle Abgrenzung gegeneinander. Das MVC-Design ist in Abbildung 5.3 als Ausschnitt der MapBiquitous-Architektur dargestellt. Das *Model* steht für alle Elemente der Software, welche Anwendungsobjekte darstellen. Sie halten die notwendigen Daten für den Programmablauf. *View-Objekte* übernehmen die Darstellung der Objekte auf der Nutzeroberfläche. In MapBiquitous sind dies die Klassen, die in Verbindung mit

dem Renderer stehen. Der *Controller* verwaltet den Datenfluss zwischen Modells und Views. Er erhält Eingaben des Nutzers und reagiert mit den entsprechenden Verarbeitungsmethoden. Der Controller beeinflusst damit wiederum die Darstellung und bildet die Eingaben auf die Modells ab.

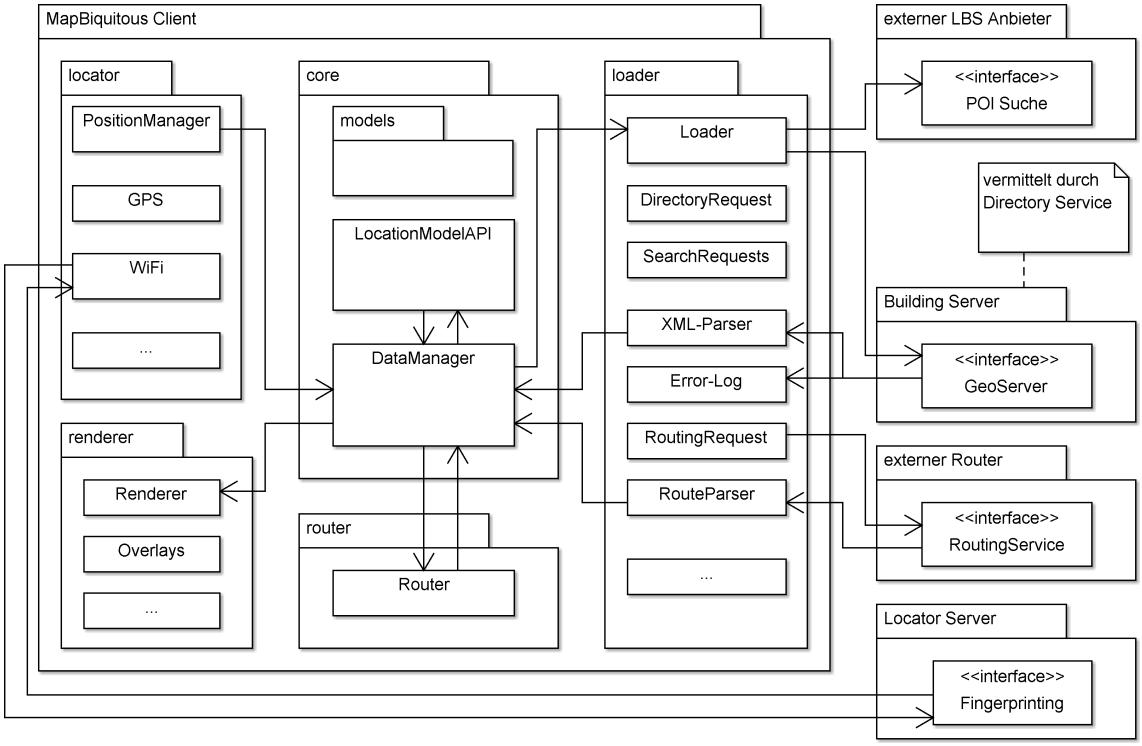


Abbildung 5.4: Package-Architektur

In Abbildung 5.4 ist der Package-orientierte MapBiquitous-Client abgebildet. Zur zentralen Komponente wird der *DataManager* innerhalb des core-Packages implementiert. Er soll als Singleton während der Initialisierung der App geladen werden und übernimmt die Verwaltung aller anfallenden Daten. Durch seine Verwaltungsfunktion kommt er im MVC-Pattern dem Controller gleich. Die Speicherung der Laufzeitdaten erfolgt derzeit bereits zum Teil in der *LocationModelAPI*. Dieses Konstrukt wird stärker an den DataManager gebunden und hält alle auf dem Client relevanten Daten bereit. In der LocationModelAPI werden die meisten der Modells, welche die Datenstruktur der Layer, Räume, u.a. abbilden, zentralisiert. Zusammen bilden sie die Model-Objekte des MVC-Pattern. Wie die in den Modells gespeicherten Daten an die unterschiedlichen Komponenten ausgegeben werden oder von ihnen aktualisiert werden, entscheidet der DataManager. Die Singleton-Implementierung (siehe [TS04], [GHVJ09]) stellt sicher, dass keine Objekte doppelt im Speicher gehalten werden. Zusätzlich dient auch der DataManager zum Cachen oder Freigeben von Speicherbereichen. In Bezug zu den anderen Programmteilen stellt er die Implementierung des Entwurfsmusters *Fassade* (siehe [TS04] und [GHVJ09]) dar. Die Fassade ist die beste Möglichkeit, in diesem Zusammenhang eine universelle Schnittstelle für die Subkomponenten des Gesamtsystems darzustellen (nach [Lar02]).

Der Locator wird in seiner Rolle als unabhängiges Positionierungsmodul bestärkt. Er implementiert das Entwurfsmuster der *Strategie*, um immer die bestmögliche Positionierung an den DataManager zu liefern. Auf diese Weise stellt der Locator in regelmäßigen Abständen die Position

fest und kann dabei auf verschiedene eigene Module zurückgreifen. Das Strategie-Muster ermöglicht, dass dem Locator unterschiedliche Algorithmen zur Verfügung stehen, um seine Aufgabe zu erfüllen (nach [GHVJ09]). Zum Beispiel kann bei vorhandenem GPS-Signal die Ortung über eben dieses erfolgen. Im Indoor-Bereich kommen Verfahren der WiFi-Lokalisierung hinzu, die eine höhere Genauigkeit liefern. Die Nutzung anderer Sensoren ist ebenfalls denkbar. Der Locator wählt zu den äußeren Gegebenheiten und verfügbaren Sensoren den besten Algorithmus und ermittelt so die Position. Wenn die aktuellen Koordinaten erfolgreich an den DataManager gesendet wurden, stehen sie somit auch allen anderen Komponenten zur Verfügung. Der Locator hat je nach Verfahren auch die Möglichkeit, mit anderen Servern zu kommunizieren.

Der Renderer registriert sich ebenfalls beim DataManager, um stets über Positions- und Datenupdates informiert zu werden. Liegen diese vor, werden die nötigen Daten an den Renderer gesendet. Er kann den Kartenausschnitt für den Nutzer erstellen und die ausgewählten Layer anzeigen. Der Renderer übernimmt damit alle darstellungsrelevanten Funktionen und bedient im MVC-Pattern das View-Konzept.

Für den Fall, dass neue Daten zum Darstellen der Gebäude oder zum Ausführen von Suchanfragen geladen werden müssen, übernimmt der Loader die Kommunikation mit dem Gebäudeserver. Die Anfrage nach den fehlenden Daten wird durch den DataManager ausgelöst. Dies ist der Fall, wenn noch keine Gebäudedaten vorliegen, diese aber dargestellt werden müssen. Es kann ebenfalls eintreten, dass eine Suchanfrage gesendet wird und die Ergebnisse noch nicht vorliegen. Sie müssen geladen werden und via Parser in Objekte überführt werden. Wenn die Suche bereits ausgeführt wurde und sich an den Parametern nichts geändert hat, muss die Anfrage beim erneuten Anzeigen der Ergebnisse nicht neu geladen werden. Um die Requests auszuführen, stehen spezialisierte Klassen im Loader zur Verfügung. Sie werden je nach Notwendigkeit vom Loader erstellt. Dieses Vorgehen entspricht dem *Factory*-Entwurfsmuster ([GHVJ09]). Die Factory oder auch Fabrik ist ein Erzeugungsmuster. Es wird genutzt, um in Abhängigkeit des zu lösenden Problems, das passende Objekt zu erzeugen. Die unterschiedlichen Request-Klassen, welche erzeugt werden können, werden in Kapitel 5.2.3 genauer erläutert. Bei der Erzeugung des Requests wird ebenfalls entschieden, welche Art Ergebnis erwartet wird. Das heißt, dass ein Parser-Objekt erzeugt wird, welches die Serverantwort analysieren und in ein Objekt überführen kann. Die erzeugten Ergebnisobjekte werden als Antwort des Loaders an den DataManager geliefert. Der Loader ist mit diesem Aufbau flexibel an alle Schnittstellen des Gebäudeservers anpassbar. Es ist auch möglich, externe Anbieter auf diese Weise einzubinden und je einen Parser zur Ergebnisauswertung zu definieren. Die Fehlerauswertung wird für den Fall ausgeführt, dass das Parsen oder der Request fehl schlägt. Die Requests und die Parser dieser Architektur implementieren das *Adapter*-Entwurfsmuster ([GHVJ09]). Dieses Muster stellt die Verbindung zwischen zwei unterschiedlichen Schnittstellen dar. Im beschriebenen Fall ist es die RESTful API des Servers, die mit Requests bedient werden muss und XML-Dateien als Responses sendet, sowie die Objektmodell-Architektur des Clients. Der Adapter transformiert die Daten zwischen beiden Seiten.

Die letzte Komponente des Frameworks ist der Router. Er dient zur Navigation und Routenberechnung. Wie alle anderen Komponenten ist er eng mit dem DataManager verknüpft und erhält über ihn die Daten. Die Berechnung der Routen innerhalb von Gebäuden erfolgt über einen Routing Service. Zusätzlich besteht die Möglichkeit, dass Routen gebäudeübergreifend berechnet

werden. Da zur Outdoor-Navigation verschiedene Anbieter zur Verfügung stehen, sollten diese genutzt werden. Der Router greift auf einen der externen Routing Services zu und nutzt vorhandene Funktionalitäten. Sobald ihm alle Teilrouten vorliegen - die innerhalb und die außerhalb von Gebäuden - setzt er sie zusammen und übergibt sie dem DataManager.

Die Entwicklung weiterer Systemkomponenten ist durch die beschriebene Architektur vereinfacht. Das System kann daher stets effektiv weiter entwickelt und an aktuelle Forschungsschwerpunkte angepasst werden.

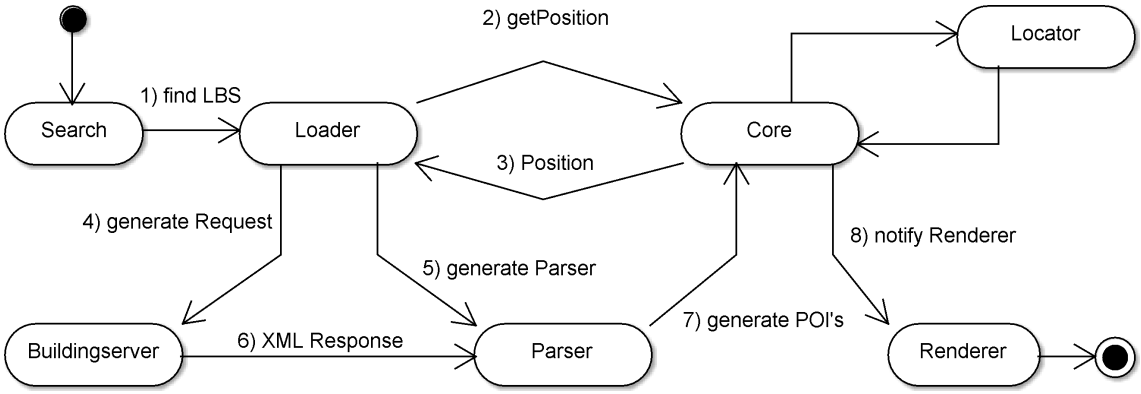


Abbildung 5.5: Anwendungsfall

Ein Anwendungsfall soll die komplexe, paketorientierte Architektur für ein besseres Verständnis beschreiben (dargestellt in Abbildung 5.5): Der Nutzer stellt eine Suchanfrage nach den umgebenden, verfügbaren Location-Based Services (Punkt 1 der Abb.). Dazu ist als erstes die genaue aktuelle Position des Nutzers notwendig (Punkt 2 der Abb.). Wenn keine aktuelle Position vom DataManager verwaltet wird, wird der Locator angewiesen eine Position zu bestimmen. Darauf folgend werden die Suchparameter mit der Position und der Art der Anfrage vom Loader an den Server gesendet (Punkt 4 der Abb.). Zeitgleich zum generierten asynchronen Request wird der Parser für das erwartete Ergebnis registriert. Der Parser erwartet den Response und erhält so wie in Punkt 6 direkt vom Server die Daten. Er analysiert anschließend die erhaltene XML. Ist er erfolgreich, werden die Informationen in POI's umgewandelt und ein Layer mit den gefundenen Ergebnissen generiert. Dieses Layer wird dem DataManager übergeben (Punkt 7 der Abb.). Der Renderer erhält eine Notification über die geänderte Schicht und stellt diese auf der Karte dar. Die Anfrage des Nutzers ist somit bearbeitet und die Ergebnisse sind sichtbar in der Karte markiert.

5.2.2 Konfiguration des Servers

Auf den Gebäudeservern läuft aktuell die Version 5.2.1 (siehe Kapitel 2.4) des MapServers. Das aktuelle Release 6.0.1[1] bietet zwar weitere Fixes im Umgang mit XML-Dateien und einige neue Features, ist aber weiterhin nicht transaktional. Da das Implementieren der WFS-T-Funktionalitäten laut offizieller Roadmap auch nicht geplant ist[2], muss für die Auslieferung der Geodaten eine Alternative gefunden werden (Anforderung A1.3). Die Entscheidung fiel auf den GeoServer in der

[1] `http://mapserver.org/` (Zugriff am 11.12.2011)
[2] `http://mapserver.org/ogc/wfs_server.html` (Zugriff am 10.01.2012)

Version 2.1.3[3]. Der GeoServer gilt als Referenzimplementierung und unterstützt damit alle standardisierten Features in der Geodatenverarbeitung. Darunter auch die entscheidenden: WSF, WSF-T und das Filter Encoding.

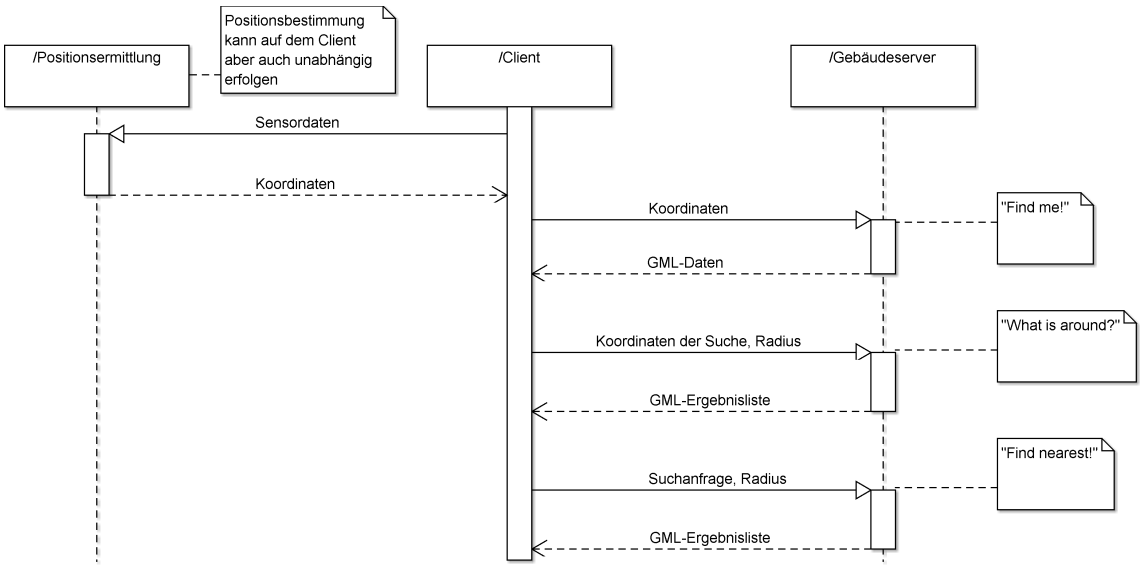


Abbildung 5.6: Feature Request an den Server und Response

5.2.3 Kommunikation mit dem Server

Die Kommunikation mit dem Server geht prinzipiell vom Client aus. Zu Serveranfragen kommt es, wenn Aufgaben gelöst werden müssen, die mit den aktuellen Client-Daten nicht gelöst werden können. Die Funktionen aus Abbildung 5.6 gehen aus der Analyse von Kapitel 4.2 hervor und sind die wichtigsten Anwendungen für Serveranfragen. Die Anforderungen, die dies betrifft sind: A2.1 bis A2.7 und A2.9. Wenn die Sensoren des Clients nicht zur Positionsbestimmung herangezogen werden können, ist dies die erste und grundlegende Funktionalität (Anforderung A2.2), welche in Zusammenhang mit einem Server gelöst werden kann. Als Beispiel dafür ist die Ortung mit Hilfe des Fingerprintingverfahrens zu erwähnen.

Datenabfragen sind auf drei unterschiedliche Arten möglich. Die Gebäudeserver dienen dabei dem Client als „Lieferant“ der verschiedensten Geodaten. Ein Client-Request wird erfolgreich durch das Ausliefern von Geodaten, kodiert in GML, beantwortet. Der zuerst beschriebene „Find me!“-Request beruht auf der Anforderung A2.7. Dabei geht es darum, dass der Client als Parameter seinen ermittelten Standort zum Server sendet. Die Responsedaten, die er erhält, sind Daten zu seinem aktuellen Standort und dem Gebäude in dem er sich befindet.

Der zweite Request („What is around?“) beschreibt die Vorgehensweise für A2.6. Die gesendeten Koordinaten können der eigene Standort oder ein auf der Karte gewählter Ort sein. Ein weiterer Parameter ist der Radius im welchem gesucht werden soll. Der Server filtert nach die-

[3] `http://geoserver.org/display/GEOS/Welcome` (Zugriff am 10.01.2012)

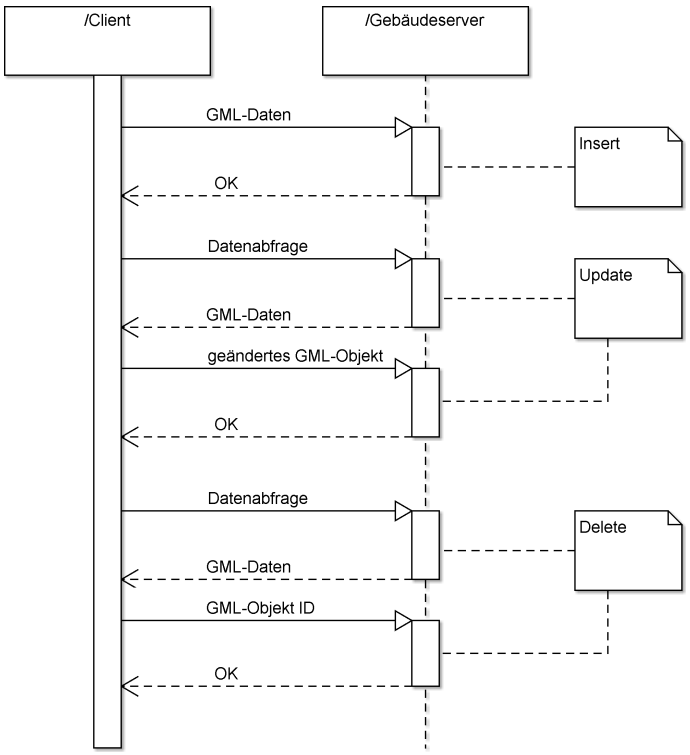


Abbildung 5.7: Feature editieren und an den Server senden

sen Kriterien den Datenbestand und sendet die erhaltenen Ergebnisse in Form einer GML-Liste an den Client zurück.

Für den dritten Anfragefall „Find nearest!“ steht Anforderung A2.4 und A2.5. Vorausgesetzte Parameter sind ebenfalls die aktuellen Standortkoordinaten und der Radius. Diesmal zählen nur die Koordinaten des Nutzers, da die nächstmögliche Einrichtung gefunden werden soll. Zusätzlich werden Suchparameter mitgesendet nach denen der Server filtert. Die erhaltenen POI's werden ebenfalls als GML-Liste an den Client geliefert. Der Filter des Servers arbeitet nach den Spezifikationen des WFS aus Kapitel 2.3.4.

Datenpflege wie sie in den Anforderungen A2.3 und A2.9 gefordert ist, muss in drei Fälle aufgeteilt werden. In allen drei Varianten findet die Kommunikation erneut zwischen Client und Server statt (siehe Abbildung 5.7). Zum Einfügen von noch nicht vorhandenen Daten („Insert“) sendet der Client die GML-Daten des neu zu erstellenden Geoobjektes an den Gebäudeserver. Dieser validiert die erhaltenen Daten. Ist die Validierung erfolgreich, wird auf dem Server das Geoobjekt mit einer eindeutigen ID, einem Zeitstempel und den gelieferten Daten erstellt. Der Status der Transaktion wird an den Client zurück gesendet.

Die zweite Transaktion besteht darin, dass vorhandene Geoobjekte geändert werden können, das sogenannte „Update“. Hierzu sind zwei Anfragen des Clients nötig. In der ersten fragt der Client die Ressource vom Server ab. Wenn er die Daten fehlerfrei erhalten hat, kann er daraus die ID des Geoobjektes isolieren. Alle Attribute, welche aktualisiert werden sollen, müssen zusammen mit der ID als GML-Objekt kodiert und an den Gebäudeserver gesendet werden. Wieder führt der

Server eine Validierung mit einer anschließenden Versionierung durch. Durch die Versionierung werden Nutzer und Zeitstempel mit dem neuen Datensatz verknüpft und machen alle Änderungen an einem Geoobjekt nachvollziehbar. Wenn der Prozess fehlerfrei abläuft, wird der OK-Status an den Client gesendet.

Ähnlich läuft die dritte mögliche Aktion („Delete") ab. Nach der Abfrage der Geodaten wird die ID inklusive dem Befehl zum Löschen des Objekts an den Gebäudeserver gesendet. Die Statusmeldung über den Erfolg der Transaktion wird wiederum an den Client gesendet und kann ausgewertet werden.

5.3 DARSTELLUNG

Die Anforderungsanalyse hat eine Reihe an Funktionen ergeben, die für die Umsetzung auf grafischer Ebene von Bedeutung sind. Sie sind im Abschnitt 4.3.1 aufgelistet. Dabei kommt der Kartendarstellung eine besonders große Bedeutung zu. Alle Aktionen, welche der Nutzer vornehmen kann und Darstellungen, mit denen er interagieren kann, werden über die Karte erreichbar sein. Die Grundvoraussetzung der Anforderung A3.2 ist daher mit dem Einbinden von Karteninformationen im Hauptansicht der Anwendung erfüllt. Mit einem Screen oder einer Ansicht, wie sie in diesem Kapitel beschrieben wird, ist eine Nutzeransicht gemeint. Die Applikation enthält mehrere verschiedene Ansichten, welche den Programmstatus für den Nutzer präsentieren. Zwischen den Screens kann gewechselt werden, wenn zum Beispiel statt der Kartendarstellung ein Eingabeformular visualisiert werden soll. Im aktuell vorliegenden Prototyp hat sich die Google Maps Karte als bevorzugte Kartenansicht bewährt. Sie hat im Vergleich zu OpenStreetMap ein besseres Zoomverhalten und eine höhere Darstellungsgenauigkeit im Satelitenmodus erreicht. Alle zusätzlichen Informationen werden als Layer auf dieser Karte gerendert. Layer können beliebig hinzugefügt und entfernt werden, wobei der Gebäudegrundriss ab der festgelegten Zoomstufe immer sichtbar ist. Er bildet damit die Orientierungshilfe und Grundlage zum Darstellen weiterer POI's für den Nutzer.

5.3.1 Interaktionsmöglichkeiten

Wie in der Einführung des Kapitels bereits erwähnt, gilt es, die meisten der Funktionen für den Anwender ausgehend von der initialen Kartenansicht umzusetzen. Aus diesem Grund ist in der Abbildung 5.8 die Kartendarstellung zentraler Teil des Zustandsdiagramms. Nach jeder Interaktion des Nutzers muss die Karte neu gerendert werden, um die abgefragten Daten in einem Layer darzustellen. Die Abbildung enthält, konform zu den Anforderung A3.6 und A3.7, die zu nutzenden Design Pattern als Aktivitäten nach [Tod11] und [VWW10].

Zoomen wird in der Karte durch *double tap* (Abbildung 5.9a), *pinch* (Abbildung 5.9b) oder *spread* (Abbildung 5.9c) ermöglicht. Der *tap* (Abbildung 5.10a) ist das kurze Berühren des Screens, double tap demzufolge zwei aufeinander folgende kurze Berührungen. Pich bezeichnet das Zusammenziehen eines Rechtecks mit zwei Fingern, wohingegen spread das Gegenteil bedeutet. Beim

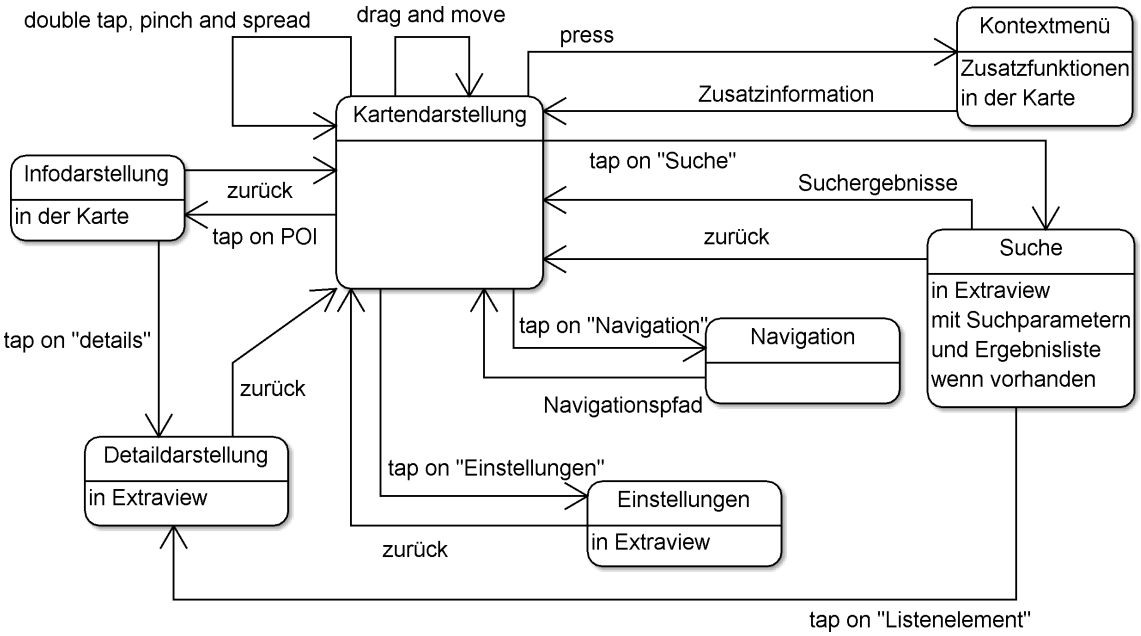


Abbildung 5.8: mögliche Nutzerinteraktionen in MapBiquitous

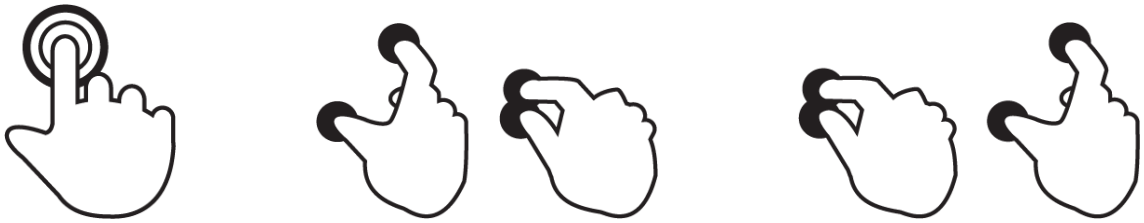
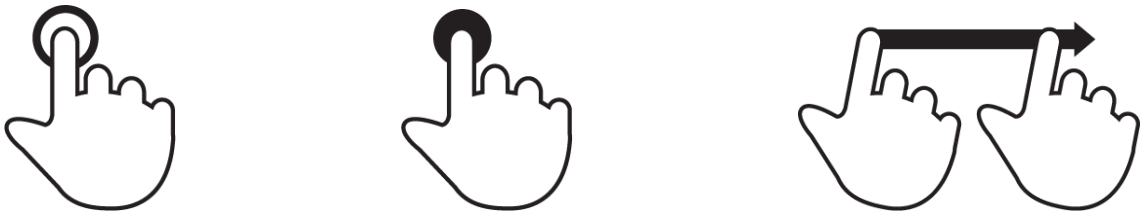


Abbildung 5.9: Touchscreen-Pattern aus [VWW10]

double tap wird die Karte immer vergrößert. Die beiden anderen Aktionen können den Kartenausschnitt vergrößern (spread) oder verkleinern (pinch). Beim Vergrößern der Karte (kleinerer Ausschnitt) wird das Anfragen neuer Daten nur notwendig, wenn ein neues „Level of Detail" erreicht wird. Das kann passieren, wenn ein Zoomlevel erreicht wird, welches das Einblenden genauerer Informationen erlaubt, oder die Kartenkacheln in einer höheren Auflösung nachgeladen werden müssen. Das Verkleinern der Karte fordert in den meisten Fällen ein Nachladen von Kartendaten, da bisher nicht geladene Kartenausschnitte visualisiert werden müssen.

Verschieben des Kartenausschnittes geschieht mit *drag and move* (Abbildung 5.10c). Der Nutzer kann durch einfaches „Anfassen und ziehen" den Kartenausschnitt ändern. In diesem Anwendungsfall müssen ebenfalls die neu angezeigten Kartenausschnitte nachgeladen werden. Sofern POI's oder Suchergebnisse bereits für den Nutzer angezeigt werden, müssen diese ebenfalls auf ihre Position überprüft und eventuell nachgeladen werden.

Suchen ist eine der Hauptfunktionalitäten, die dem Nutzer Extrainformationen bereitstellen. Um die Aktion auszulösen gibt es zwei verschiedene Möglichkeiten: Die erste interaktive Variante



(a) tap (b) press (c) drag and move

Abbildung 5.10: Touchscreen-Pattern aus [VWW10]

besteht darin, das *press*-Pattern (nach [VWW10] siehe Abbildung 5.10b) anzuwenden. Es beruht darauf, dass der Nutzer einen Punkt in der Karte längere Zeit mit dem Finger berührt. Nach dem entscheidenden Zeitraum wird für den berührten Kartenpunkt die passende Information nachgeladen. Anschließend wird die Info direkt in der Karte dem Punkt zugehörig eingeblendet. Der press ermöglicht somit das Hinzufügen eines Kontextmenüs mit weiteren Funktionen und Suchmöglichkeiten an einen bestimmten Kartenpunkt.

Die zweite Suchmöglichkeit ist die direkte Suche. Über das Menü erreicht der Nutzer den Screen, der ihm eine Suchmaske zur Verfügung stellt. Die Parameter, wie zum Beispiel der Suchterm werden mit *Soft keyboard* in ein *Inputfeld* eingetragen. Das ausgefüllte Formular wird an den Server gesendet. Die Antwort enthält die Geodaten, welche wiederum in der Karte dargestellt werden. Wurde die Suche erfolgreich ausgeführt, existiert neben der Kartendarstellung mit eingezeichneten Ergebnissen auch eine Listenansicht. Diese kann angezeigt werden, wenn der Nutzer das Suchformular erneut aufruft. Er sieht die Parameter, welche zur aktuellen Suche geführt haben und nachfolgend die Ergebnisse als scrollbare Liste. Bei der Wahl eines Ergebnisses öffnet sich die Detailansicht des Ergebnisses. Über den Zurück-Button gelangt man wieder zur Kartenansicht mit eingeblendeten Ergebnissen.

Navigation ist ein weiterer Menüpunkt. Über ihn erreicht der Nutzer ein Formular, das die Eingabe von Start und Ziel ermöglicht. Um die Berechnung zu starten, drückt der Nutzer den OK-Button und gelangt zur Kartenansicht mit den Navigationshinweisen.

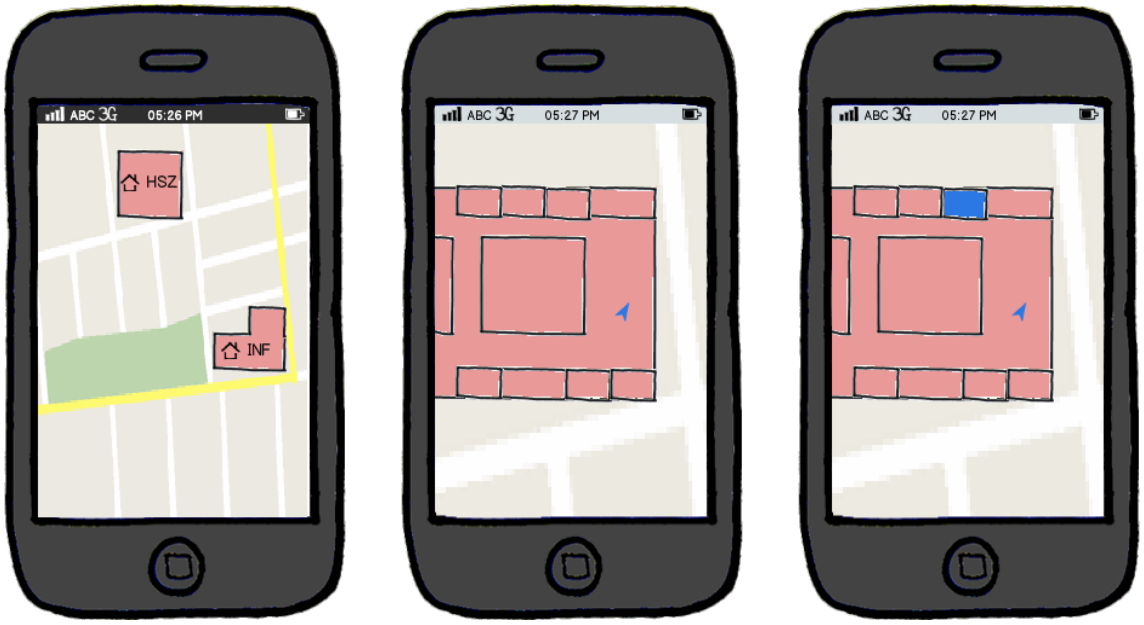
Einstellungen heißt der letzte Menüpunkt. Er führt zu einem Screen, welcher die Personalisierung der Application ermöglicht. Damit wird die Anforderung A3.8 an einer zentralen Stelle der Anwendung erfüllt. Nach dem Tätigen der Einstellungen wird der Menüpunkt durch Drücken der Zurück-Taste wieder verlassen. Es erscheint erneut die Kartenansicht.

Kurzinfos und Details lassen sich anzeigen, indem auf einen POI oder ein Suchergebnis getippt wird. Dabei wird zuerst eine Zusatzinformation direkt am Punkt in der Karte eingeblendet. Als weiterer Schritt ist es möglich, für den entsprechenden Punkt weitere Detailinformationen anzufordern. Dazu gelangt man auf einen neuen Screen, der diese Metainformationen darstellt. Der Zurück-Button ermöglicht zu jeder Zeit den Wechsel zur Kartenansicht. Das gleiche Verhalten wird beim Auswählen eines Raumes angestrebt. Wenn die Berührung des Displays auf die

Koordinaten eines eingeblendeten Raumes oder Gebäudeumrisses zurückgeführt werden kann, sollen die verfügbaren Metadaten des Objektes eingeblendet werden.

5.3.2 Mockups

Um die in Abschnitt 5.3.1 beschriebenen Interaktionen zu verdeutlichen, werden in diesem Kapitel Mockups genutzt. Mockups sind Skizzen der Nutzeroberfläche. Auf ihnen ist die Struktur der Anwendung erkennbar. Sie dienen nicht der Beschreibung des exakten Designs, sondern der Verdeutlichung einzelner Szenarien. Die bereits beschriebenen Möglichkeiten werden so bildlich untermalt und können dadurch schneller verstanden und umgesetzt werden.



(a) Ansicht mit Gebäudegrundrissen

(b) Raumansicht bei höherer Zoomstufe

(c) Auswahl eines Raums

Abbildung 5.11: Mockups zur Kartenansicht

Die Kartenansicht wurde als Einstiegspunkt für den Nutzer gewählt und ist in Abbildung 5.11a schematisch dargestellt. Man erkennt die Gebäudeumrisse eingebettet in der Karte. Zusätzlich können die Grundrisse beschriftet sein, um eine Grundinformation für den Nutzer bereit zu stellen. Das Zoomlevel entscheidet dabei ab welcher Größe die Umrisse eingeblendet werden, da es wenig Sinn macht, beim Rendern eines Landes die einzelnen Gebäude einzublenden. Das bisherige Zoomlevel ist 16. Die Stufe wurde gewählt, da bei einem höherer Zoom bereits deutlich erkennbare Grundrisse gezeichnet werden können.

Gleiches gilt für das nächste Detaillevel. Sobald die Zoomstufe für das Anzeigen einzelner Räume erreicht ist, werden diese innerhalb des bereits vorhandenen Grundrisses gerendert (Abbildung 5.11b). Wenn die Position des Nutzers auf Grund der Einstellungen und der vorhandenen Sensoren erkannt werden kann, wird der Standort mit einem Pfeil angezeigt. Durch diesen Hinweis

gelingt die Orientierung innerhalb des Gebäudes. Der Pfeil zeigt, sofern es möglich ist, auch die Orientierung in Bezug auf die Blickrichtung an.

Sobald die Räume erkennbar sind, können diese per tap gewählt werden (Mockup 5.11c). Sie erscheinen danach markiert und es können Kurzinformationen angezeigt werden. Diese haben die gleiche Darstellung und den gleichen Inhalt wie die bereits beschriebene „Infodarstellung". Beispielhaft ist das Verhalten in Abbildung 5.13a erkennbar und wird im Abschnitt des Suchens genauer beschrieben.

(a) eingeblendetes Menü

(b) Einstellungen in einer Extraansicht

(c) eingeblendete Suchmaske

Abbildung 5.12: Mockups zu Menü, Einstellungen und Suche

Menü und Einstellungen dienen dem Nutzer zum Zugriff auf weitere Ansichten und Funktionen. Das Menü ist über den „Menü-Button" des mobilen Endgerätes erreichbar. Es wird eingeblendet und ermöglicht dem Nutzer die Navigation zu weiteren Screens und Formularen (Abbildung 5.12a). Die drei beispielhaft gezeigten Einträge für „Suche", „Navigation" und „Einstellungen" können durch weitere einfach erweitert werden. Dazu sollte bei unzureichendem Platz eine zweite Zeile eingeführt werden. Um den Platzverbrauch des Menüs zu beschränken sind alle weiteren Menüpunkte in einer Liste unterzubringen, welche über einen Menüeintrag mit der Bezeichnung „mehr..." zu erreichen ist.

Der Screen, in dem die personalisierten Einstellungen vorgenommen werden können, ist im Mockup 5.12b dargestellt. Die Einstellungen sind listenartig untereinander angeordnet. Es existiert bei Bedarf eine Beschreibung, die als solche durch Farb- und Größenabstufungen erkennbar ist. Für die Einstellungen stehen verschiedene Möglichkeiten zur Verfügung:

Checkboxen, die eine zusätzliche Funktion aktivieren

Dropdowns, mit denen zwischen Standardwerten gewählt werden kann

Buttons, welche auf einen weiteren Screen verweisen, um komplexere Einstellungen vorzunehmen

Optiongroups, werden eingesetzt, wenn eine übersichtliche Zahl von Auswahlmöglichkeiten dargestellt werden muss

Die Suche kann der Nutzer wie erwähnt über das Menü erreichen. Er erhält dann ein Formular (Abbildung 5.12c), welches alle Parameter der Suche aufnimmt. Darin werden die freien Suchterme des Nutzers per Softkeyboard eingegeben. Standardwerte aus den Einstellungen werden angezeigt, können aber auch bei Bedarf geändert werden. Über Buttons lässt sich die Suche starten oder abbrechen.

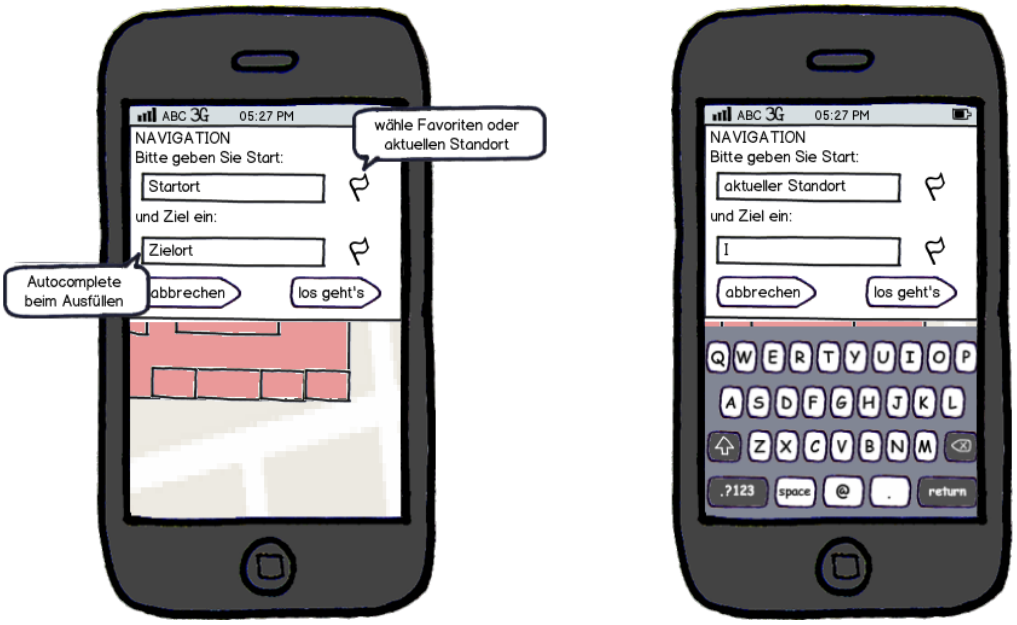
(a) Ansicht der Gebäudegrundrissen mit Suchergebnissen

(b) Suchergebnisse als Liste

(c) Details eines POI bzw. Suchergebnisses

Abbildung 5.13: Mockups zur Karten-, Listen- und Detailansicht

Die Ergebnisse der Suche können auf verschiedene Art und Weise angezeigt werden. Initial wird dazu, wie in Mockup 5.13a, die Anzahl der Ergebnisse für den Nutzer dargestellt. Zusätzlich werden alle Ergebnisse, die im aktuellen Kartenausschnitt darstellbar sind, angezeigt. Der tap auf ein solches Ergebnis, welches durch ein spezielles Icon repräsentiert wird, zeigt dem Nutzer die Infodarstellung. Sofern weitere Details gefragt sind, kann der Nutzer entweder die Detaildarstellung direkt am Ergebnis aufrufen und gelangt auf eine Ansicht ähnlich Abbildung 5.13c oder er ruft zuerst die Liste der Suchergebnisse auf (Abbildung 5.13b). Diese erlangt er mit einem erneuten tap auf die Suchfunktion. In der Listenansicht kann er sich dann ein Ergebnis wählen und gelangt ebenfalls zur Detailansicht. Er kann auch die Parameter, welche zur Suche führten ändern und damit die Suche erneut starten.



(a) Formular zum Starten der Navigation

(b) Navigationsformular mit Softkeyboard

Abbildung 5.14: Mockups zur Kartenansicht

Die Navigation ist ähnlich der Suche über das Menü erreichbar. Da es sich hierbei ebenfalls um ein Formular handelt, welches Start und Ziel der Navigation aufnehmen muss, ist die Darstellung ebenfalls an die Suche angelehnt (siehe Abbildung 5.14a und 5.14b). Zusätzlich besitzt jedes der beiden Texteingabefelder einen Button, der den Nutzer vordefinierte Standorte durch einen einfachen tap auswählen lässt. Dazu zählt zum Beispiel der aktuelle Standort, sofern dieser ermittelt werden kann, eine Liste von Favoriten, auf welche zurückgegriffen werden kann und eine Möglichkeit zur direkten Auswahl eines Ortes im Gebäude oder auf der Karte. Zusätzliche Nutzerfreundlichkeit schafft die Autocompletefunktion, welche beim Schreiben in die Textfelder ausgelöst wird.

Wenn in beiden Feldern ein gültiger Standort eingetragen ist, kann die Berechnung zur Navigation durch einen „Start“-Button angestoßen werden. Das Ergebnis der Berechnung wird nachfolgend in die Kartenansicht gezeichnet. Die Navigation kann dann gestartet werden.

Die Dateneingabe wurde bereits aus architektonischer Sicht im Kapitel 5.2.3 beschrieben. Um die Schnittstellen zu bedienen und damit die Datenbestände aktualisieren zu können, ist eine Eingabeoberfläche für den Nutzer notwendig (siehe Anforderung A3.1). Für das Bearbeiten bestehender POI's wird ein Menüpunkt angelegt, der zu einem Formular mit möglichen POI-Eigenschaften in Form von Eingabefeldern weiterleitet. Diese Felder sind mit den aktuellen Werten gefüllt und können vom Nutzer korrigiert werden. Ein ähnliches unausgefülltes Formular erhält der Anwender, wenn ein neuer POI angelegt werden soll. Bei diesem Szenario muss er zusätzlich die Position durch einen tap auf die Karte wählen können. Alle Informationen werden nach erfolgreicher Validierung an den Server gesendet. Die Antwort des Servers bestätigt den Erfolg der Aktion und beendet das Erstellen bzw. Bearbeiten.

Das Datenformular wird so flexibel gestaltet, dass auch andere Eingabemöglichkeiten als Erweiterung denkbar sind. Dazu zählen zum Beispiel die Datenerfassung via QR-Code oder die Möglichkeit, einen Fingerprint der aktuellen Sensordaten als Geotag einzutragen.

5.3.3 Farbgestaltung

Bei der Farbgestaltung muss einerseits auf eine ausreichende Kontrastwirkung zwischen Kartendarstellung und gerenderten Gebäuden geachtet werden, andererseits sollten sich alle Farben harmonisch in das Gesamtbild der App einfügen. Um diese Konsistenz auch für Erweiterungen des Frameworks zu erhalten, werden die Farben und ihr Anwendungsbereich in der Konfiguration der Anwendung global definiert (Anforderung A1.8 und A3.8). Der Entwickler greift dann beim Zeichnen eines Objektes auf die entsprechende Farbkonstante zu. Die Auslagerung der Farbdefinition erlaubt ebenfalls eine schnelle und unkomplizierte Anpassung des kompletten Farbschemas.

Bezeichnung	Hex-Farb-Code	Erklärung
Grundriss	#FF9367	der Grundriss des Gebäudes im LOD2
Grundrissumrandung	#FF533C	die Umrandung des Grundrisses
Innenhof	#FCF6A4	der Innenhof eines Gebäudes
Raum	#FF9367	die Fläche eines Raumes im LOD3
Raumumrandung	keine	die Umrandung eines Raumes
ausgewählter Raum	#005C60	ein vom Nutzer ausgewählter Raum
Umrandung des ausgewählten Raums	keine	die Umrandung des ausgewählten Raumes
Sonderraum	#8374B5	ein speziell hervorgehobener Raum (z.B. WC's)
Umrandung des Sonderraums	keine	die Umrandung spezieller Räume
POI-Markierung	#8374B5	die Markierung eines Suchergebnisses
Standortmarkierung	#05A1CD	die Markierung des Nutzerstandortes
Route	#89ECFF	der Navigationspfad
Start der Route	#98E79A	ein ausgewählter Startraum oder Startpunkt
Ziel der Route	#FF9099	ein ausgewählter Zielraum oder Zielpunkt

Tabelle 5.1: Farbdefinition für zeichenbare Objekte

Die geplanten Funktionen fordern Farbdefinitionen für verschiedene Bereiche, welche in der Tabelle 5.1 inklusive einer Erklärung aufgelistet sind. Die Bereiche haben sich aus der Untersuchung der aktuellen App und einer unterstützenden Evaluation ergeben. Der entwickelte Fragebogen ist im Anhang A.1 zu sehen. Er ist so konzipiert, dass zu jeder Frage mehrere Bilder mit einer vierstufigen Skala bewertet werden sollten. Diese sogenannte Likert-Skala (nach [Lik32]) wurde bewusst auf eine gerade Anzahl von Antworten beschränkt, um den Probanden zu einer positiven oder negativen Bewertung zu drängen. Eine neutrale Enthaltung der Testpersonen war daher ausgeschlossen. Die Farbvarianten, über die entschieden werden musste, sind im An-

hang A.2 dargestellt. Jedes einzelne Bild wurde etwa 30 Sekunden an eine Leinwand projiziert. Diese Zeit war ausreichend, um die Bewertung im Fragebogen anzukreuzen. Dieses Testverfahren wird als *AB Test* bezeichnet. Der AB Test wie er in [HPSG11] vorgeschlagen wird, dient in der Entwicklungs- und Konzeptionsphase dem Vergleich verschiedener Produktversionen. Bei der Evaluation wurden insgesamt 28 Studenten in der Vorlesung „Application Development for Mobile and Ubiquitous Computing“ mit Hilfe dieses Tests befragt. Durch den Vorlesungshintergrund waren alle Befragten mit dem Thema der Anwendungsentwicklung speziell auf mobilen Geräten und der Konzeption von Nutzeroberflächen vertraut. Sie stellten daher ideale Testpersonen dar. Die Reaktionen der Probanden auf unterschiedliche Versionen einer Darstellung wurden erfasst und ausgewertet. Die Designvorschläge stellten in diesem Fall jeweils verschiedene Farbgebungen von Grundrissen und Raumvisualisierungen dar (siehe Anhang A.1 und A.2).

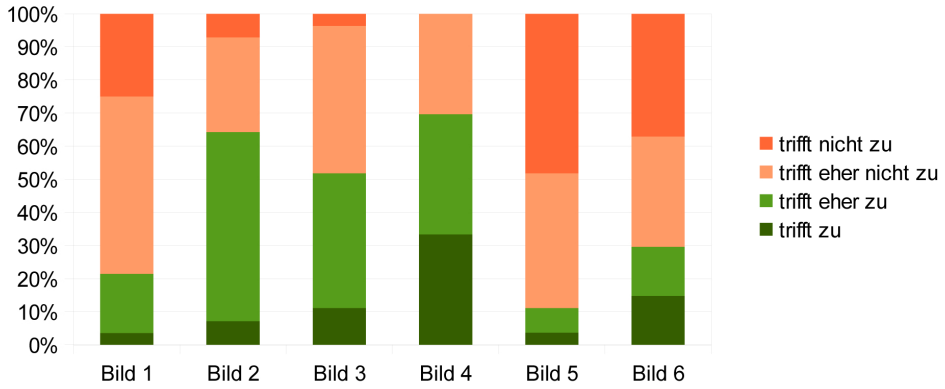
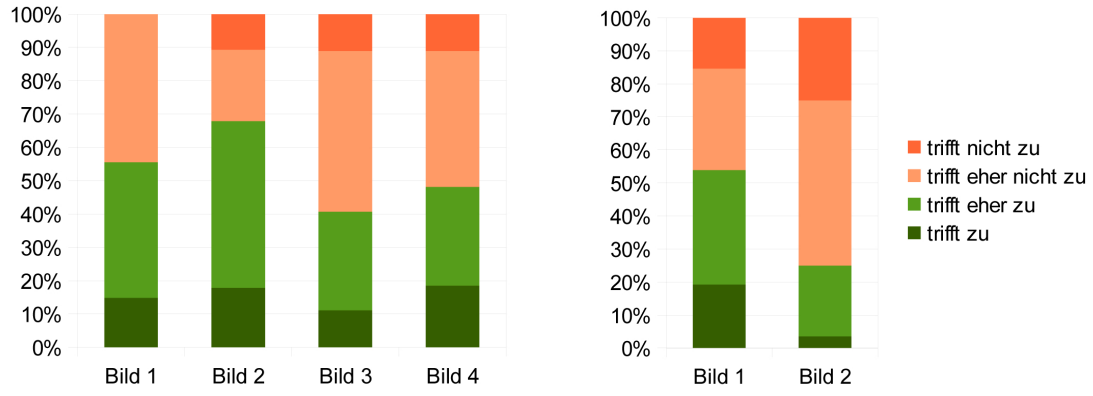


Abbildung 5.15: Diagramm - Auswertung Frage 2



(a) Auswertung Frage 3

(b) Auswertung Frage 4

Abbildung 5.16: Diagramme der Farbschemaevaluation

Die Ergebnisse der Evaluation sind in der folgenden Auflistung ausformuliert und in der Tabelle 5.1 konkret festgelegt.

Grundrisse sind deutlicher erkennbar, wenn ein Umriss im selben Farbton aber geringerem Hel-

ligkeitsanteil und höherer Sättigung die eigentliche Gebäudefläche umrahmt. Diese Meinung wird im Auswertungsdiagramm 5.15 deutlich. 33% vertraten diese Ansicht voll, 36% sagten, dass diese Aussage „eher zutrifft".

Der Innenhof passt sich am besten in das Gesamtbild ein, wenn er heller und mit weniger Sättigung als das Gebäude dargestellt wird. Ein extra Umriss für den Innenhof ist nicht notwendig.

Sonderräume sollten mit Kontrastfarben zu den übrigen Räumen dargestellt werden bzw. müssen eine deutliche Abstufung zu den anliegenden Farben aufweisen. Umrandungen sind in der Darstellung der Räume nicht förderlich. Diese Schlussfolgerung lässt sich aus der Auswertung der dritten Frage ziehen. Die entsprechende Farbkombination in Bild 2 wurde von 18% („trifft zu") und 50% („trifft eher zu") der Befragten als die beste gewählt (siehe Diagramm 5.16a).

Ausgewählte Räume sollten genau wie Sonderräume durch Kontraste hervorgehoben werden. Diese Entscheidung teilten etwas mehr als die Hälfte (54%) aller Testpersonen (Auswertungsdiagramm 5.16b).

5.4 ZUSAMMENFASSUNG

Am Ende des Kapitels werden die wichtigsten Punkte der Konzeption zusammengefasst und die nächsten Schritte erläutert. Dadurch sollen die konzipierten Änderungen verdeutlicht werden.

Der Abschnitt 5.1 beschreibt die nötigen Änderungen an der Datenstruktur. Es wurde deutlich gezeigt, dass sowohl auf Datenbankseite, als auch auf dem Client zusätzliche Tabellen bzw. Modells geschaffen werden müssen. Die Erweiterungen werden nötig, um die neuen Informationen eines POI's abbilden zu können. Es wurde festgelegt, was die zusätzlichen Elemente erfüllen müssen. Dazu zählt unter anderem die Vererbungsstruktur zwischen dem neuen PointOfInterest und dem LocatableItem, welche auf dem Client die beste Flexibilität für die neue Klasse garantiert.

Für die Architektur des Clients wurde die Frameworkstruktur mit Hilfe von Designpattern aus der Softwareentwicklung beschrieben (Abschnitt 5.2). Das MVC-Pattern ist dabei das beste Mittel, um die einzelnen Komponenten des Frameworks zu trennen. Die Aufspaltung der Funktionsbereiche in Model-View-Controller-Packages ist die wichtigste Änderung des Frameworks. Hinzu kommen weitere Pattern, welche die Implementierung erleichtern, da sie dem Framework logische Strukturen vorschreiben. Der Aufbau des Frameworks wurde an Hand von neu konzipierten Packages erklärt und in verschiedenen Abbildungen verdeutlicht. Die neuen Funktionen für Serverkommunikation und Einbindung weiterer LBS wurden so beschrieben, dass die Umsetzungen klar definiert sind. Dazu zählt hauptsächlich der neu zu konfigurierende Gebäudeserver in Form des Geoservers und die damit entstehenden Möglichkeiten zur Datenabfrage und Datenpflege.

Im letzten Abschnitt (5.3 Darstellung) wurde das neue Konzept für die Nutzeroberfläche vorgestellt. Hier wurden gebräuchliche Interaktionsmuster und deren Einsatzgebiete beschrieben. Es

wurde festgelegt, welche Funktionen über welche Menüstrukturen und Nutzerinteraktionen erreichbar sein sollen. Für die Nutzeransichten wurden Mockups erarbeitet. Sie stellen die möglichen Ansichten dar und geben klare Vorgaben für die konzipierte Umsetzung. Abschließend wurde das Farbschema der Anwendung durch eine Nutzerbefragung definiert. Die dadurch erarbeitete Farbgebung aller möglichen Designelemente wurde detailliert aufgelistet.

Um das Konzept in seiner vorgestellten Version zu testen, muss es im folgenden Kapitel (6 Implementierung) umgesetzt werden. Die reine Konzeption erfolgte ohne Spezialisierung auf eine bestimmte Plattform. Die Umsetzung und Implementierung wird mit Android erfolgen, da der vorhandene MapBiquitous-Prototyp eine gut geeignete Ausgangsbasis hierfür bietet. Anschließend wird zusätzlich eine Nutzerevaluierung durchgeführt. Sie dient der Überprüfung der entworfenen Konzepte auf der Nutzeroberfläche.

6 IMPLEMENTIERUNG

Das folgende Kapitel beschreibt ausführlich die implementierten Änderungen am MapBiquitous-Prototypen. Die praktische Umsetzung der im Kapitel 5 erarbeiteten Konzepte ist aus Entwicklersicht die beste Variante eben diese zu testen und zu evaluieren. Um einen schlüssigen Vergleich zwischen der Konzeptstruktur und den verwirklichten Funktionen zu ziehen, ist das Kapitel ähnlich gegliedert. Zu Beginn wird die Neukonfiguration der Gebäudeserver und die sich dadurch ändernde Datenstruktur beschrieben. Anschließend werden die geänderte Architektur und die implementierten Schnittstellen erläutert. Der dritte Abschnitt behandelt die Änderungen der Nutzeroberfläche und somit das angepasste Bedienkonzept. Den Abschluss des Kapitels bilden die aufgetretenen Probleme und weitere Änderungen, die bisher noch nicht eingeordnet wurden.

6.1 DATENSTRUKTUR

Das Konzept sieht in den Abschnitten 5.1.1 und 5.1.2 die Anpassung der vorhandenen PostGIS-Datenbank und die Erweiterung der client-seitigen Modellstruktur vor. Die weitreichenden Veränderungen auf beiden Seiten der Architektur sollten das Zusammenspiel zwischen Client und Server jedoch nicht beeinträchtigen. Die gewünschten zusätzlichen Funktionalitäten forderten Veränderungen im Programmablauf sowie in der grundlegenden Datenbereitstellung, welche im Folgenden aufeinander aufbauend beschrieben werden.

6.1.1 Datenbank

Als Grundlage für weiterführende LBS und zum „Befüllen“ der Modells des Clients wurde in Kapitel 5.1.1 der Point of Interest als „Datencontainer“ eingeführt. Die Datenpunkte sollten ebenfalls auf den Gebäudeservern abrufbar sein und für den Nutzer wichtige Services und Informationen innerhalb und außerhalb von Gebäuden zur Verfügung stellen. Um diese neuen Daten zu speichern, wurde die Erweiterung der Datenbank als beste Möglichkeit herausgestellt. Bereits in

den „Related Works“ (Kapitel 3) wurden notwendige Attribute zum Speichern der Informationen untersucht und anhand der gewonnen Ergebnisse diskutiert. Im Ergebnis dieser Überlegungen ist ein neuer Tabelleneintrag in der PostGIS-Datenbank konzipiert und umgesetzt worden. Die entstandene Tabelle „poi“ enthält folgende Daten:

name - stellt den Anzeigenamen des POI's dar. Er dient dem Nutzer zur ersten Einordnung und groben Beschreibung. Es handelt sich dabei um eine String-Repräsentation in der Datenbank.

type - ist ein Integer, der den POI in eine Kategorie einordnet. Die Kategorisierung findet derzeit client-seitig statt. Dabei wird dem Integer ein Kategoriename zugeordnet. Die Entscheidung, einen Typ für jeden POI einzuführen ist auf die Untersuchung in Kapitel 3.1 zurückzuführen.

description - beschreibt den POI detaillierter und stellt weitere Informationen für die Anzeige des Punktes bereit. Dieser Wert wird ebenfalls als String in der DB abgelegt.

floor - ist die zweistellige Zeichenkette, welche die Etage repräsentiert oder *null*, wenn der POI außerhalb eines Gebäudes ist. Mit dieser Information lassen sich POI's auf der Nutzeroberfläche etagenweise anzeigen.

geb_id - ähnlich der floor-Eigenschaft, dient die geb_id (Integer) dazu, einen Verweis auf ein Gebäude zu speichern. Dabei handelt es sich um eine ID, die als Fremdschlüsselbeziehung genutzt werden kann.

geometry - codiert einen *gml:Point* nach OpenGIS, der zum Speichern der Koordinaten des POI verwendet wird.

last_update - stellt ein Datum dar, welches die letzte Aktualisierung des POI's speichert. So kann die Aktualität der Informationen überprüft werden. Als Zeitstempel wird die Unixzeit nach dem Standard [Com08] verwendet.

create_date - stellt das Erstelldatum des Punktes dar. Auch bei diesem Attribut wird die Unixzeit nach dem Standard [Com08] verwendet.

uid - bietet eine eindeutig identifizierbare ID. Sie kann verwendet werden, um den POI unabhängig sämtlicher anderer Voraussetzungen referenzieren zu können. Die UUID wird nach dem Standard [Fou05] gebildet und als String gespeichert.

gid - stellt eine Geo-ID der PostGIS-Datenbank dar. Sie wird automatisch durch PostGIS erzeugt und verwaltet. Die gid, dient dem Aktualisieren oder Löschen der Geodaten.

Alle einzufügenden Zeichenketten sind in *UTF-8* kodiert. Das ist allgemeiner Standard und erleichtert den Datenaustausch und die Darstellung auf dem Client. Die Geodaten der vorhandenen Räume und Gebäude wurden bisher im Koordinatenformat *EPSG:4326* mit dem zugehörigen geodätischen Referenzsystem *WGS 84* angezeigt. Folglich wird diese Vereinbarung auch bei den neu hinzugekommenen POI's eingehalten.

Einige Testdatensätze wurden direkt mit QuantumGIS 1.6.0 in die erstellte Datenbanktabelle eingefügt. Die Vorgehensweise sei hier nur der Vollständigkeit halber kurz erwähnt:

Der erste Schritt ist die Einbindung vorhandener Geodaten in QuantumGIS, um eine grundlegende Orientierung zu erhalten. Die Daten können aus der PostGIS-Datenbank oder aus Shape-Dateien geladen und dargestellt werden. Als nächstes sollte die neu erstellte „poi"-Tabelle geladen und zur Bearbeitung markiert werden. Nun können mit dem Zeichenwerkzeug Punkte in das neue Layer gesetzt werden. Jeder Punkt hat die in der Tabelle angegebenen Attribute. Diese sollten einzeln vervollständigt werden. Zum Schluss der Bearbeitung werden die Daten direkt in die PostGIS-DB gespeichert und stehen ab sofort als Ausgangsdaten zur Verfügung.

6.1.2 Serverkonfiguration

In der Konzeption (Abschnitt 5.2.2) wurde der Geoserver als Server zur Verwaltung der Geodaten festgelegt. Dieser existierte bisher nicht und musste an die vorhandene und bereits erweiterte Datenbank angebunden werden. Der Geoserver wurde in der Version 2.1.3 im Tomcat des carlos-Servers[1] deployed. Er ist somit unter der Url `http://carlos.inf.tu-dresden.de:8080/geoserver/` für den Client erreichbar. Da dieser Server neu aufgesetzt wurde und die Funktion der bisherigen Gebäudeserver übernehmen sollte, musste die Einrichtung von Grund auf erfolgen. Die Konfiguration spiegelt die Architektur des Gesamtsystems wieder und wird deshalb in den nächsten Paragrafen Schritt für Schritt beschrieben:

Arbeitsbereiche (engl. workspaces) werden vom Geoserver verwendet um verschiedene Namensräume (engl. namespaces) voneinander zu trennen. Im MapBiquitous-Projekt wird diese Möglichkeit genutzt, um unterschiedliche Gebäudeserver zu simulieren und über jeweils eindeutige Urls anzusprechen. Die Konvention, pro Gebäude einen Gebäudeserver und damit einen Namensraum zu nutzen, unterstützt den Gedanken der Skalierbarkeit. Das System ist damit ohne Weiteres auf beliebig viele Gebäude erweiterbar. Im Fall des aktuellen Datenbestandes wurden fünf Namensräume angelegt: *tud_bar, tud_hsz, tud_inf, tud_m13 und tud_poi*. Um die dadurch entstehenden virtuellen Gebäudeserver zu erreichen, bildet man die Url nach dem Muster: `http://carlos.inf.tu-dresden.de:8080/geoserver/[namespace]/`.

Datenquellen (engl. stores) stellen die Geodaten für den Geoserver zur Verfügung. Um die Idee der Namensräume konsequent zu verfolgen, wird für jeden Namensraum eine Datenquelle angelegt. Eine Datenquelle stellt im MapBiquitous-Projekt jeweils alle Daten für ein Gebäude zur Verfügung. Das heißt, die Datenquelle ist die bereits besprochene PostGIS-Datenbank.

Layer (engl. Layers) dienen zum Darstellen der Daten einer Etage oder eines Grundrisses. Ein Layer erhält im Geoserver alle seine Daten aus einer Datenquelle. Diese Daten können, da es sich bei den definierten Datenquellen um Datenbanken handelt, mit SQL-Befehlen eingeschränkt und

[1] `http://carlos.inf.tu-dresden.de`

vorselektiert werden. Durch diese Abfrage werden ebenfalls die Attribute des Layers festgelegt. Zusätzliche Metainformationen wie das Koordinatenreferenzsystem, die geografische Ausdehnung des Layers und der Typ der Geodaten müssen angegeben werden, um die problemlose Bearbeitung der Geodaten zu gewährleisten.

Am Beispiel der Informatikfakultät wurde also ein Arbeitsbereich mit zugehöriger Datenquelle angelegt. Weiterhin wurden fünf Etagen (inklusive Keller) durch je einen Layer dargestellt. Für vier der Etagen existieren Positionen der WLAN-Accesspoints, die zusätzlich als je ein Layer eingefügt wurden. Zuletzt wurde für die allgemeine Ansicht des Gebäudes der Grundriss als Layer im Geoserver konfiguriert. Alle aufgezählten Layer haben, da sie ein Gebäude repräsentieren, den selben Namensraum.

Der WFS des Geoservers muss angepasst werden, um die gesamte Funktionalität freizuschalten. An dieser Stelle werden Einstellungen festgelegt, die für die Verarbeitung der Daten auf dem Client entscheidend sind. So wird die Kodierung der XML und die Interpretation der GML-Versionen bereits auf dem Server eingestellt. Zusätzlich werden über das „Service-Level" die transaktionalen Fähigkeiten des WFS aktiviert. Diese werden später zum Einfügen und Ändern von Geodaten benötigt.

Globale Einstellungen existieren ebenfalls. Sie legen die Zeichenkodierung und die Genauigkeit der ausgegebenen Koordinaten fest. Ersteres sollte konsistent zur DB-Einstellung, UTF-8, sein. Die Nachkommastellen der Koordinaten wurden auf Grund der bisherigen client-seitigen Projekteinstellung auf sechs Stellen festgelegt.

Directory Service Neben dem Geoserver existiert der Directory Service (DS). Er reagiert auf Anfragen des Clients und verteilt die Urls der Gebäudeserver. Da sich die Urls auf Grund des neu eingerichteten Geoservers geändert haben und auch die Namensräume eingeführt wurden, ist es notwendig, die neuen Daten in den DS einzupflegen. Die bisherigen Namen der Gebäude erhalten den Namensraum als Präfix in der Form: *[namespace]:[building name]*. Die Url zum WFS ändert sich auf die beschriebene Adresse der Form: `http://carlos.inf.tu-dresden.de:8080/geoserver/[namespace]/wfs?&SERVICE=wfs&VERSION=1.0.0`. Durch diese Anpassungen sind die Gebäudedaten wie bisher über den DS erreichbar. Die ausgelieferten Adressen werden auf dem Client genutzt, um über den WFS die Layer-Daten abzufragen und schließlich dem Nutzer zu visualisieren.

6.1.3 Modellstruktur

Eine der Hauptzielstellungen der Arbeit ist die Einführung einer stabilen und effektiven Frameworkstruktur. In der Konzeption aus Kapitel 5.2 wurden dafür eine Reihe an Designpattern und Implementierungskonzepte vorgestellt und angewandt. Dabei ist deutlich geworden, dass die Kapselung der Aufgabenbereiche eine besonders wichtige Rolle spielt. Die Trennung verschiedener Funktionen ermöglicht erst die gewünschte Erweiterbarkeit und Flexibilität des entstandenen

Frameworks. Das MVC-Pattern hat sich bei der Untersuchung und anschließenden Konzeption als beste Möglichkeit zum Gliedern des Frameworks erwiesen. In diesem Abschnitt wird erläutert, wie es im aktuellen Prototypen zur Anwendung kommt und welche Veränderungen sich daraus ergeben.

Packages sind in Java ein gängiges Mittel, um durch unterschiedliche Namensräume Javaklassen voneinander zu trennen. Ein Package kann gesondert komprimiert werden und auch in anderen Projekten als Bibliothek importiert werden. Üblicher Weise verbindet die Klassen eines Packages eine ähnliche Funktionalität oder die gleiche logische Einteilung. Sie sind damit genau das Mittel, um Funktionalitäten im MapBiquitous-Framework festzulegen.

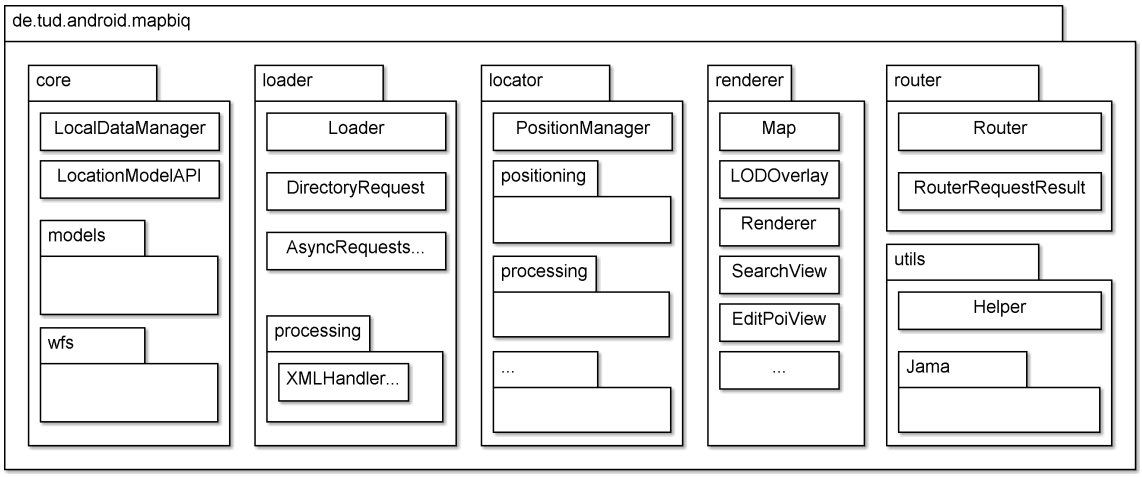


Abbildung 6.1: die wichtigsten Packages mit ausgewählten Klassen

Der Namensraum der gesamten Anwendung ist **de.tud.android.mapbiq**. Alle Programmelemente befinden sich damit innerhalb des Haupt-Packages „mapbiq". Die Abbildung 6.1 zeigt die wichtigsten Packages des MapBiquitous-Frameworks. Es wird nur eine Auswahl der wichtigsten dargestellt und auch wenige Klassen eingezeichnet. Das Haupt-Package teilt sich, wie grundlegend im Kapitel 5.2.1 beschrieben, in den *core*, den *loader*, den *locator*, den *renderer* und den *router* auf. Zusätzlich ist ein Package mit dem Namen *utils* entstanden. Die ehemalige Anwendungsstruktur ist damit hinfällig und nicht mehr vorhanden. Neu hinzukommende Anwendungskomponenten sollten nach dem beschriebenen Konzept in die vorhandene Struktur und in die passenden Packages eingeordnet werden. Dieses strikte Vorgehen erhält die Übersichtlichkeit des stetig wachsenden Projektes. Eine Ausnahme bilden Komponenten, die thematisch nicht verortbar sind. Dieser Fall kann durchaus eintreten, wenn das Framework um spezielle Aufgaben erweitert wird, die mit dem heutigen Stand der Technik und Entwicklung noch nicht absehbar waren. Eine solche Erweiterung würde seine Funktionalitäten in einem eigens angelegten neuen Package bündeln.

Um die Änderungen zu verdeutlichen, werden die genannten Packages kurz beschrieben, ihre enthaltenen Funktionalitäten genannt und wenn nötig die Herkunft ihrer Elemente geschildert.

Das core-Package bildet den Hauptbestandteil der App. Direkt im core findet die *LocationModelAPI* und der *LocalDataManager* seinen Platz. Die LocationModelAPI hat die Funktion, den Großteil der Daten zu speichern. Werden aufwendigere Verarbeitungsmechanismen oder Zugriffsmethoden benötigt, finden sich diese im LocalDataManager wieder. So wird zum Beispiel das Caching über den LocalDataManager abgewickelt. Diese beiden Klassen speichern und verwalten damit den Kern der Anwendung. Andere Komponenten können über sie die Modells und Informationen für den Programmablauf anfordern. Des Weiteren sind unter *core.models* alle Klassen aus Abschnitt 5.1.2 zusammengefasst. Sie dienen der Abbildung von Daten und werden von anderen Klassen initialisiert. Ähnliches gilt für die Klassen aus *core.wfs*. Diese Klassen bilden die WFS-Services und Layer ab und sind über die LocationModelAPI erreichbar.

Der loader-Package kapselt alle Funktionalitäten zur Kommunikation mit dem Geoserver und anderen externen Diensten. Er enthält überwiegend asynchrone Tasks, welche über die *Loader*-Klasse initialisiert und ausgeführt werden. Die Loader-Klasse wird im Androidprototypen als *Service* implementiert und kann so auch Intents anderer Komponenten abfangen und darauf reagieren. Ist ein Request erfolgreich, wird der vom Loader generierte Callback ausgeführt und stellt das Ergebnis der LocationModelAPI bzw. dem LocalDataManager zur Verfügung. Um die asynchronen Tasks und deren Ergebnisse auszuwerten, besitzt jeder Task einen Callback, der aus *loader.processing* den entsprechenden Ergebnishandler aufruft. Diese Handler sind zu großen Teilen XML-Parser, welche die Responses überprüfen und in eine Modellstruktur überführen. Der *DirectoryRequest* wurde ebenfalls in den loader aufgenommen, da er mit einem externen Server kommuniziert. Zum Überprüfen des Erweiterungskonzeptes wurden nachfolgend aufgelisteten Request-Klassen implementiert und getestet. Die jeweilig zugehörigen Parser werden nicht extra aufgeführt, sind aber gleichermaßen erfolgreich umgesetzt.

AsyncGeoserverSearchTask - er dient zur etagenübergreifenden Suche in einem Gebäude. Dabei werden die in Kapitel 2.3.5 erläuterten Filtermechanismen des WFS angewendet.

AsyncGoogleSearchTask - ist ein Aufruf der Google Places API, welcher Suchergebnisse in einem definierten Radius ausgibt (siehe 3.2.3).

AsyncLBSTask - stellt die Überklasse zu den hier beschriebenen Requests dar. Er gibt die Struktur und der erweiternden Location-Based Tasks vor. Wenn Requests an externe Dienstleister und an interne Gebäudeserver von diesem Task erben, wird sichergestellt, dass die erhaltenen Ergebnisse einheitlich verarbeitet und dargestellt werden können.

AsyncOpenStreetMapSearchTask - fragt ähnlich dem AsyncGoogleSearchTask Suchergebnisse von externen Servern ab (siehe 3.6).

AsyncPoiSearchTask - sucht nach POI's, welche nicht über den Gebäudeserver erreichbar sind.

Das locator-Package stellt für die Anwendung eine weitere grundlegende Funktionalität zur Verfügung - die Positionierung des Nutzers. Da die Sensoren, welche zu einer genauen Indoor-Position führen können, immer zahlreicher werden, ist der locator in weitere Unter-Packages aufgeteilt. Dazu zählen unter anderem: *locator.processing.fingerprinting* oder *locator.processing.wlan*.

Diese Komponente der Anwendung ist zur Zeit noch in der Entwicklung, um die bestmögliche Positionierung zu untersuchen. Sie hat aber bereits ihre finale Kommunikationsschnittstelle mit dem core. Sobald eine neue Position verfügbar ist, wird diese in der *LocationModelAPI* gesichert und ist über *getPosition* auslesbar. Dadurch wird auch an dieser Stelle des Projektes deutlich, dass die Zusammenarbeit der unterschiedlichen Komponenten durch Neuentwicklungen und Erweiterungen durch die neue Struktur nicht blockiert wird.

Das renderer-Package ist für alle Ausgaben an den Nutzer der App verantwortlich. Im renderer-Package wurden alle Klassen zusammengefasst, die mit dem Zeichnen von Overlays auf die Kartenansicht oder anderen Views in Verbindung stehen. Damit ist die komplette Ausgabe auf dem Screen über den renderer zu verwalten. Nutzereingaben werden ebenfalls vom renderer entgegen genommen und dann via Intents oder durch direktes Speichern in den core an andere Programmkomponenten weitergereicht. Da es notwendig wurde die Darstellung anzupassen, haben sich viele Änderungen in der *LODOverlay*-Klasse ergeben. Außerdem sind die beiden Views *EditPoiView* und *SearchView* hinzugekommen. Mit ihnen können POI's angelegt und geändert bzw. gesucht werden. Sie dienen vorrangig dazu, dem Nutzer die nötigen Eingabemöglichkeiten zur Verfügung zu stellen. Weitere Details im Hinblick auf die angepasste Darstellung werden im Kapitel 6.3 erläutert.

Das router-Package ist ähnlich wie der locator eine Komponente, an der parallel zu dieser Arbeit geforscht und gearbeitet wird. Sie wurde komplett neu in das umstrukturierte Framework eingefügt. Dieser Programmteil stellt einen weiteren Location-Based Service zur Verfügung. Es handelt sich dabei um die integrierte Indoor- und Outdoor-Navigation. Start- und Zieldaten werden nach dem Konzept dieser Arbeit bereits über den core verwaltet und mit Hilfe des Renderers dargestellt. Auch dieses Package unterstützt daher die Kapselung von Funktionen. Es greift ebenfalls auf den loader zu, um eine Schnittstelle zu externen Routingservices zu etablieren.

Das utils-Package bietet eine Menge an hilfreichen Funktionen. In dieses Package wurden Funktionalitäten ausgelagert, die von verschiedenen anderen Klassen verwendet werden. Die Auslagerung in ein extra geschaffenes Package hat den Vorteil, dass der bestehende Code von doppelten Algorithmen befreit wurde. Änderungen an grundsätzlichen Berechnungen sind damit nur noch an einer einzigen Stelle im Projekt notwendig und ziehen sich nicht mehr durch viele Klassen hindurch. Dieses Vorgehen vermindert auch die Fehleranfälligkeit des Codes im Allgemeinen. Zu den utils gehören überwiegend geometrische Berechnungen, die Inklusionstests und ähnliches durchführen. Es sind ebenfalls Umrechnungsmethoden für Punktkoordinaten und standardisierte Algorithmen in dieser Komponente zu finden.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- add and change here all settings to configure the project -->
    <bool name="use_building_contoures">true</bool>

    <!-- positioning settings -->
```

```
    <bool name="use_sensor_positioning">false</bool>
    <bool name="dummy_positioning">true</bool>

    <!-- general search options -->
    <item name="searchRadius" type="integer">500</item>

    <!-- options for searching the geoserver, have to be available in
        all layers -->
    <string-array name="searchPropertiesGeoserver">
        <!-- the room number -->
        <item >id</item>
        <item >nutzung</item>
    </string-array>

    <!-- options for searching with google places -->
    <string name="googlePlacesApiKey">insert your Google Places key
        here</string>

    ... weitere Konfigurationen ...
</resources>
```

Code 6.1: Ausschnitt aus der config.xml

Ressourcen sind in jedem Android-Projekt unabhängig des Javacodes definiert. Zu solchen Ressourcen zählen unter anderem alle verwendeten Bilder und Icons, die Layouttemplates, welche für die Darstellung der Views benötigt werden und sämtliche Konstanten und Texte. Diese Umsetzung wird in allen androidbasierten Developer Guides als Paradigma festgelegt, wurde aber im bisherigen Prototypen kaum eingehalten. Bei der Strukturierung des Frameworks wurden diese Gegebenheiten beachtet und beispielsweise vorhandene Strings an die entsprechende Stelle ausgelagert. Die Ressourcen bieten auch die Möglichkeit mit einfachsten Mitteln Konstanten projektweit zu definieren. Da dies auch eine der Anforderungen war, sind Einstellungsmöglichkeiten in der *config.xml* vereinheitlicht und zentral zusammengefasst. Im Codebeispiel 6.1 ist ein Ausschnitt dieser Konfigurationsdatei dargestellt. Er soll zeigen wie verschiedene Datentypen definiert werden. Einfache Datentypen wie Strings, Booleans und Integer stehen dabei zur Verfügung. Auch String-Arrays sind einfach in Form von Listen definierbar. Mit diesen XML-Definitionen sind alle benötigten Konstanten abbildbar. Für weitere Typen steht die Androiddokumentation zur Verfügung[2].

[2] `http://developer.android.com/guide/topics/resources/available-resources.html` (Zugriff am 01.02.2012)

6.2 SCHNITTSTELLEN

Da das Konzept so angelegt ist, dass die Kommunikation zur Datenabfrage über den Loader gesteuert wird, wurden einige der geschaffenen Änderungen bereits im Abschnitt 6.1.3 erwähnt. Da die Änderungen für die gesamte App entscheidend und auch in der Konzeption einen besonderen Stellenwert haben, wird an dieser Stelle die beispielhafte Implementierung noch genauer erläutert. Es werden auch Codeausschnitte gezeigt an welchen das Vorgehen nachvollzogen werden kann. So soll auch für weitere Entwicklungen der Leitfaden verdeutlicht werden.

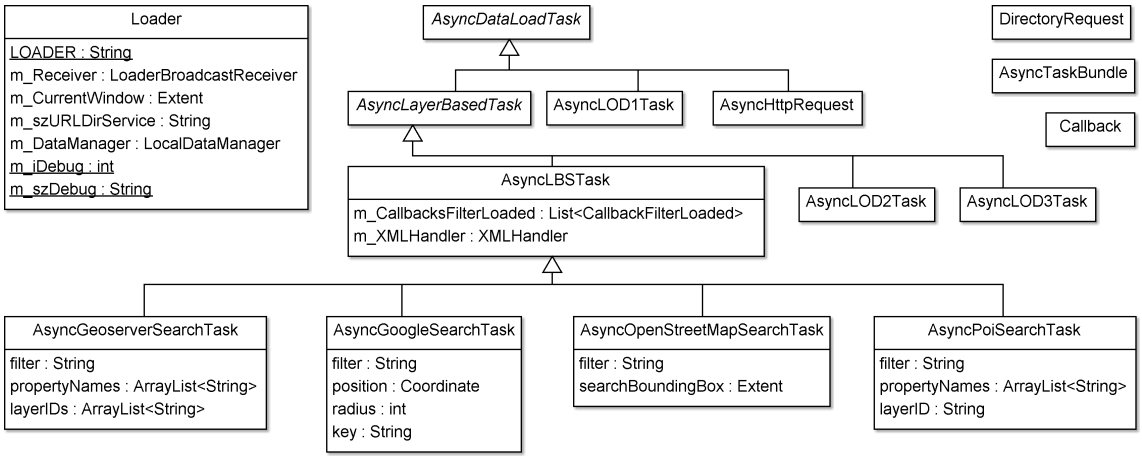


Abbildung 6.2: loader-Package mit den asynchronen Requests (Ausschnitt)

6.2.1 Beispiel „Find nearest!“

In Kapitel 5.2.3 wurden die drei hauptsächlichen Location-Based Services und deren Kommunikation mit den Servern beschrieben. Die Kommunikationswege und das Bilden der Ergebnismenge wird in diesem Abschnitt am Beispiel des stationären LBS *Find nearest!* (siehe Grundlagenkapitel 2.2.2) gezeigt. Durch die vier implementierten Requests, ist die Integration von internen und externen LBS im MapBiquitous-Prototypen umgesetzt.

Für die Suchanfrage öffnet der Nutzer die *SearchView* der App und gibt den Suchterm und das zu durchsuchende Gebäude an. Im Hintergrund werden zusätzlich Positionsdaten ermittelt. Alle Parameter werden in der LocationModelAPI gesichert um sie auch zu einem späteren Zeitpunkt wieder abfragen zu können. Beim Start der Suche setzt die Activity einen Intent mit dem Type *onGetSearchResult* ab. Wenn dieser vom Loader empfangen wird, beginnt das Initialisieren der asynchronen Requests (siehe Abbildung 6.2). Die vier Tasks für die Suche im Gebäude, nach POI's, über Google Places und auf OpenStreetMap werden vom Loader in einem Bundle von Requests zusammengefasst. Dieses Bundle erhält einen zusätzlichen Callback um die Fertigstellung aller Requests zu prüfen. Anschließend wird das Bundle mit dem *execute()*-Aufruf alle Tasks soweit möglich parallel ausführen. Jeder einzelne Task durchläuft beim Aufruf seines Konstruktors folgende Schritte:

Zuerst werden die mitgegebenen Suchparameter ausgewertet und angelegt. Danach wird der

festgelegte Handler für die Ergebnisdaten dem Callback mitgegeben. Über die Elternklasse (*AsyncLBSTask* in Abbildung 6.2) wird die Suche als „aktuell ausgeführte Suche“ im core registriert. Der Task selber startet beim Aufruf der *executeTask()*-Methode. Hierbei wird die Url für den Request generiert und geprüft. Zusätzlich werden bei jedem Request die Parameter entweder als GET oder POST-Parameter an den Request übergeben.

```
POST http://carlos.inf.tu-dresden.de:8080/geoserver/tud_poi/wfs?&
    SERVICE=wfs&VERSION=1.0.0&REQUEST=GetFeature
```

```
<wfs:GetFeature service="WFS" version="1.0.0" outputFormat="GML2"
  xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wfs  http://schemas.
      opengis.net/wfs/1.0.0/WFS-basic.xsd">
  <wfs:Query typeName="tud_poi:poi">
    <ogc:Filter>
      <ogc:Or>
        <ogc:PropertyIsLike wildCard="*" singleChar="." escapeChar="!">
          <ogc:PropertyName>gml:name</ogc:PropertyName>
          <ogc:Literal>*Sitz*</ogc:Literal>
        </ogc:PropertyIsLike>
      </ogc:Or>
    </ogc:Filter>
  </wfs:Query>
</wfs:GetFeature>
```

Code 6.2: GetFeature-Request mit POST-Daten

Mit den beschriebenen Initialisierungen und Klassen- bzw. Methodenaufrufen sind die Requests an die jeweiligen Server gesendet worden. Am Beispiel des Geoservers stellt sich der eingehende Request so dar: Über die Url und den angegebene Namensraum ist die Datenquelle, auf welche sich die Abfrage bezieht, identifizierbar. Der Dienst ist in allen Fällen als WFS angegeben, dadurch kann der Server bereits auf das Ausgabeformat der Daten schließen. Des Weiteren spezifiziert der Parameter *REQUEST* den Typ der Anfrage. In den Fällen der Suche ist dies *GetFeature*. Da der GetFeature-request mit Filterparametern nicht übersichtlich in der Url angegeben werden kann, wird das Senden von POST-Daten genutzt. Der Request Body enthält eine valide XML-Definition mit den verfügbaren OGC-Funktionen zum Filtern der Geodaten. Im Codebeispiel 6.2 ist ein Request dargestellt, der nach POI's sucht. Im *wfs:Query*-Tag ist der Featuretype festgelegt und anschließend der *ogc:Filter* definiert. Dieses Beispiel zeigt einen Filter, der mit verschiedenen Wildcards das Attribut *gml:name* des gewählten Typs durchsucht. Ein Ergebnis ist dann gefunden, wenn der String „Sitz“ in dem Attribute vorkommt. Mehrere Vergleiche können mit der *ogc:Or*-Funktion logisch verbunden werden. Mit den gefilterten Ergebnissen erstellt der Geoserver eine *wfs:FeatureCollection* im angegebenen GML2-Format. Sie enthält die einzelnen *gml:featureMember*, welche wiederum die *tud_poi:poi*-Elemente mit ihren Attributen definieren

und darstellen. Der auf diese Weise generierte XML-Output wird an den Client zurück gesendet.

Der Callback des asynchronen Tasks nimmt die Daten entgegen und reicht sie an den festgelegten XMLHandler weiter. Dieser hat nun die Aufgabe, einen neuen *PointOfInterest* zu initialisieren. Er sollte möglichst effektiv die Metadaten des Ergebnisses auf die Attribute der Klasse abbilden und die geografische Position aus den GML-Definitionen extrahieren. Alle Daten, die nicht direkt auf Attribute abgebildet werden können, werden als *ItemAttribute* in den PointOfInterest eingefügt. So stehen sie in der späteren Darstellung als Metadaten weiterhin zur Verfügung. Hat das Parsen problemlos funktioniert, sendet der Handler die Suchergebnisse an den core und meldet sich als „aktive Suche" ab.

Wenn alle Parser durchlaufen sind, wird der Callback des Bundles aufgerufen. Die Suchergebnisse können ab diesem Zeitpunkt in der Karte und als Liste in der SearchView dargestellt werden.

6.3 DARSTELLUNG

Die Darstellung und Konzeptionierung einer nutzerfreundlichen Oberfläche nahm einen großen Teil des Kapitels 5 ein. Da bisher kaum auf die Möglichkeiten der Touch-Bedienung eingegangen wurde, wurden die üblichen Design-Pattern vorgestellt und auf die entsprechenden Anwendungsgebiete übertragen. Die Konzeption hat Möglichkeiten aufgezeigt, mit denen ein ansprechendes und leicht zu bedienendes Interface für das Framework geschaffen werden kann. Zusätzlich zu allgemein gültigen Usability-Konventionen wurde mit Hilfe von Mockups „Paper Prototyping" (siehe [HPSG11]) betrieben und eine Evaluation zur Farbgestaltung (Kapitel 5.3.3) angestrengt. Die entstandenen Konzepte wurden am Prototypen umgesetzt um auch sie auf Tauglichkeit und Nutzbarkeit prüfen zu können. Sie werden in den nachfolgenden Abschnitten beschrieben.

6.3.1 Interaktionsmöglichkeiten

Die in Kapitel 5.3.1 zur Nutzung vorgeschlagenen Interaktionsmöglichkeiten wurden auf ihre aktuelle Verwendung geprüft. Es konnte festgestellt werden, dass bereits einfache Interaktionen mit der Karte möglich waren. So funktionierten das *Zoomen*, und *drag and move* ohne weiteres Zutun. Zusätzlich existierten standardisierte Zoom-In und Zoom-Out-Buttons am unteren Rand der Kartendarstellung. Um das Konzept umzusetzen, wurden daher noch die *tap* und die *press* Touchscreen-Pattern implementiert.

Der tap auf ein Gebäude oder Gebäudeteil wurde dabei genutzt, um schnell wichtige Informationen in der Karte anzuzeigen. Zusätzlich werden dem Nutzer dadurch weitere Funktionen und Menüpunkte eröffnet. Weitere Informationen und Implementierungsdetails werden im Abschnitt 6.3.2 erläutert. Das Ergebnis eines taps ist im Screenshot 6.3a in Form des „Infoballons" zu sehen.

(a) Ballon mit Informationen zum gewählten Raum

(b) Lupe und Zoom-Buttons am Bildschirmrand

(c) Anwendungsmenü

Abbildung 6.3: drei Screenshot-Ausschnitte des Androidprototypen

Mit dem tap werden üblicherweise alle Buttons aktiviert und deren Funktion ausgelöst. Um einen schnellen und unkomplizierten Zugriff auf die Suchfunktionalität zu erhalten, wurde am unteren Kartenrand ein Button eingefügt. Dieser ist im Screenshot 6.3b als Lupe zu erkennen. Das Aktivieren des Buttons führt den Nutzer zur Suchansicht der App. Das Hinzufügen des neuen Bedienelements hat es aus gestalterischer Sicht erforderlich gemacht, auch die Standard-Zoom-Buttons optisch anzupassen.

Neben der Ansicht der Suche ist auch die Navigation als neue Funktion der App hinzugekommen. Das App-Menü wurde dahingehend erweitert, dass diese neuen Möglichkeiten für den Nutzer jederzeit über die Menütaste erreichbar sind. Das aufgeräumte, neue Menü ist in Androidanwendungen standardisiert am unteren Bildschirmrand zu finden (Screenshot 6.3c).

6.3.2 Metainformationen und Zusatzfunktionen

Der bereits kurz erwähnte „Infoballon" ist ein Overlay-Element, welches an einer bestimmten Position direkt in die Karte gezeichnet werden kann. Er zeigt sprechblasenartig auf den zu beschreibenden Punkt (siehe Screenshot 6.3a). Der Ballon ist so konfiguriert, dass er die Bildschirmbreite nicht überschreitet und sich damit sehr gut in das Kartenlayout einpasst.

Die Javaklasse *InfoBalloonOverlay* wurde entwickelt, um möglichst vielseitige Darstellungsmöglichkeiten zu bieten. Sie ist in der Abbildung 6.4 des Klassendiagramms mit allen Attributen und Operationen gezeigt. Prinzipiell wird mit Hilfe des Ballons ein Titel, ein Snippet und eine Reihe an Interaktionsschaltflächen dargestellt. Die Attribute *title* und *snippet* werden bei der Initialisierung aus dem übergebenen *Item* ausgelesen. Die Buttons, welche über die Enumeration *Buttons* festgelegt sind, lassen sich über Methodenaufrufe zum Beispiel einzeln einblenden (**setButtonVisible(btn : Buttons)**) oder auch alle ausblenden (**setAllButtonsHidden()**). Sichtbare Schaltflächen können über den Aufruf **setButtonClickListener(btn : Buttons, l : OnClickListener)** eine Funktionalität zugewiesen bekommen. Dieses Vorgehen wird im Prototypen zur Etagen- oder Raumwahl genutzt.

```
1  InfoBalloonOverlay<OverlayItem> balloonView = new InfoBalloonOverlay<
       OverlayItem>(mapView.getContext(), 0);
```

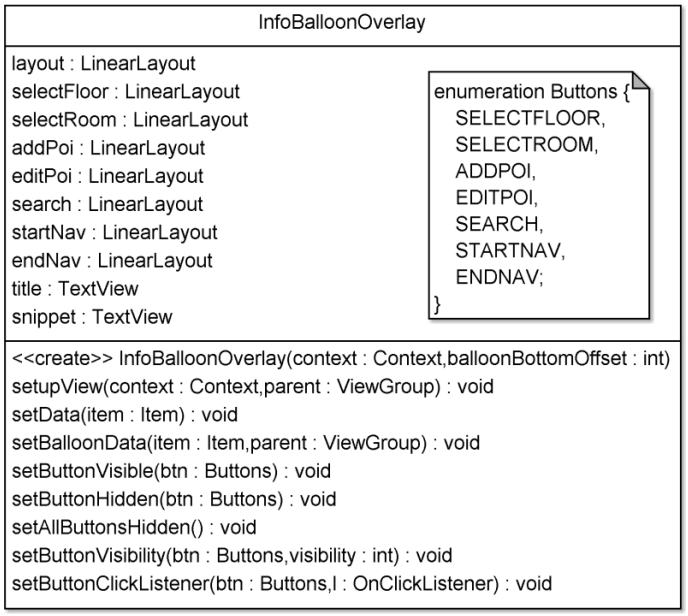
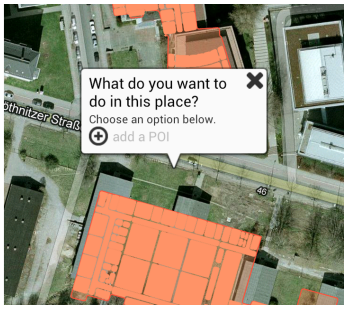


Abbildung 6.4: Klassendiagramm des InfoBallons

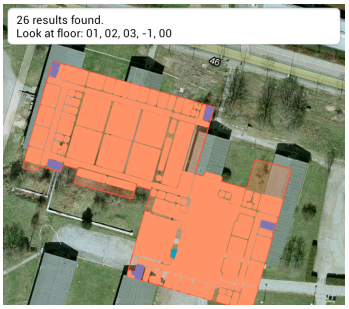
```
balloonView.setData(oItem);
balloonView.setAllButtonsHidden();
balloonView.setButtonClickListener(Buttons.SELECTFLOOR, this);
balloonView.setButtonVisible(Buttons.SELECTFLOOR);
if (floorSelected) {
  balloonView.setButtonClickListener(Buttons.SELECTROOM, this);
  balloonView.setButtonVisible(Buttons.SELECTROOM);
}
```

Code 6.3: Initialisierung und Konfiguration des InfoBallonOverlay

Das Vorgehen ist beispielhaft am Codeauszug 6.3 dargestellt. Die erste Zeile erstellt einen neuen BalloonViewOverlay, indem der Typ des Items festgelegt wird und der aktuelle Kontext dem Konstruktor übergeben wird. Die zweite Zeile setzt die darzustellenden Daten und übergibt mit dem OverlayItem gleichzeitig die Position, an welcher der Ballon gerendert werden wird. Die Zeilen drei, fünf und acht sorgen dafür, dass die gewünschten Buttons sichtbar gemacht werden und alle anderen versteckt bleiben. Buttons bestehen aus einem Icon und einem Label und werden nacheinander in einer Zeile angezeigt. Ist die maximale Ausdehnung des Ballons erreicht, bricht die Zeile, ähnlich floatenden HTML-div-Elementen, um. Das Ergebnis des Codebeispiels ist ein Ballon mit Titel, kurzer Beschreibung und einem oder wenn die if-Anweisung erfüllt ist, zwei Buttons. Die OnClickListener werden auf den OnClickListener der aktuellen Klasse umgeleitet und dort verarbeitet.



(a) Ballon mit „add POI“-Funktion



(b) Hinweis auf die Suchergebnisse

Abbildung 6.5: zwei Screenshot-Ausschnitte des Androidprototypen

6.3.3 Datenmanagement

Das *press*-Pattern, also ein länger andauernder tap, wurde genutzt, um der Anwendung weitere „Spezialfunktionen“ hinzufügen zu können. Am Beispiel des Einfügens von POI's wird die Implementierung hier erklärt. Führt der Nutzer den press auf einer Stelle der Karte aus, bekommt er den gewohnten Ballon mit Informationen zur aktuell gewählten Position. Diese sind auf Grund der Interaktion verschieden von denen, die ein tap nach sich gezogen hätte. Dem User wird dies im Text des Ballons verdeutlicht und er bekommt andere Auswahlmöglichkeiten für weitere Interaktionen. Im Beispielscreenshot 6.5a ist das die Möglichkeit, einen Point of Interest an der gewählten Stelle in die Karte einzupflegen. Ähnlich könnte verfahren werden, um einen POI zu editieren. Nimmt der Nutzer diese Möglichkeit war, wird er auf eine neue Activity weitergeleitet, welche ihm ein Formular zur Dateneingabe bereitstellt. Folgende Felder kann er ausfüllen, um den Geotag zu erstellen:

name - stellt ein einzeiliges Textfeld für die Bezeichnung des POI's dar.

type - wird als Dropdownliste dargestellt und gibt dem Nutzer in der config.xml vordefinierte Auswahlmöglichkeiten.

building - ist ein Autocomplete-Feld, in welchem die aktuell geladenen Gebäude zur Verfügung stehen.

floor - stellt die Etage dar und ist ebenfalls als Dropdown bereits vorkonfiguriert.

description - wird als mehrzeiliges Textfeld angezeigt, da der User hier eine umfassendere Beschreibung einfügen kann.

Diese Felder entsprechen der jeweiligen Repräsentation in der POI-Datenbanktabelle (siehe Abschnitt 6.1.1). Die weiteren Daten, wie die geografische Position und Datumsangaben, kann der Nutzer nicht ändern. Sie werden automatisch ermittelt.

6.3.4 Suche und Ergebnisanzeige

Über welche Schnittstellen und auf welchen Datengrundlagen die Suche ausgeführt wird, ist in den vorangegangenen Kapiteln bereits beschrieben worden. Dieser Abschnitt verdeutlicht noch einmal die neu hinzugekommenen Nutzerschnittstellen. Wie bereits erwähnt, ist die Suche direkt über einen Button in der Karte oder über das Menü erreichbar. Beide Interaktionen führen zur selben Ansicht. Sie ist ebenfalls als Formular von Eingabefeldern designt. Es handelt sich hierbei aber nur um zwei Felder. Das erste dient zum Erfassen des Suchterms, das zweite zum Festlegen des Gebäudes, welches durchsucht werden soll. Das Feld zur Gebäudewahl ist ein Autocomplete-Feld, welches bereits alle erreichbaren Gebäude geladen hält. Der Nutzer erhält daher direkt nach dem Tippen der ersten Buchstaben eine gefilterte Auswahl an Gebäuden. Der genaue Ablauf nach dem tap auf den „Search"-Button wurde in Kapitel 6.2.1 beschrieben. Dabei wurde auch erwähnt, dass die Suchparameter im core gesichert sind. Sobald die SearchView das nächste Mal geladen wird, wird sie mit den gesicherten Parametern initialisiert. Zusätzlich wird überprüft, ob Ergebnisse der letzten Suche vorliegen. Ist das der Fall, werden sie in einer Liste unter den zwei Textfeldern eingeblendet. Dadurch kann der User sich neben der Darstellung der Punkte in der Kartenansicht auch die Ergebnisse als Liste ansehen.

Für die Kartenansicht gilt, dass diese nach erfolgreicher Suche automatisch aktualisiert wird und die Ergebnisse als Punkte oder bei gefundenen Räumen als Polygone einzeichnet. Da im Inneren von Gebäuden die Suchergebnisse auf mehreren Etagen liegen können, wird ein Hinweis auf die Anzahl und die betroffenen Etagen im oberen Bereich der Karte eingeblendet (siehe Screenshot 6.5b). Sofern für das Gebäude keine Etage geladen wurde und nur der Grundriss sichtbar ist, werden alle Ergebnisse gerendert, um einen Überblick über deren Verteilung zu geben. POI's müssen nicht zwingend eine Etage enthalten. Wenn dieser Fall eintritt, werden sie immer unabhängig der geladenen Etage angezeigt.

6.3.5 Navigation

Am Beispiel der Navigation lässt sich erkennen, wie gut sich eine neue Programmkomponente mit zusätzlichen Funktionen in das Gesamtprojekt integrieren lässt. Die Navigation an sich ist nicht Teil dieser Arbeit, sie wurde aber in enger Zusammenarbeit und Planung von Anfang an berücksichtigt.

Das Grundgerüst für die Architektur und die Einbindung der Routerkomponente wurde vom Frameworkkonzept vorgegeben. Auch die Farbgebung der verschiedenen Elemente, wie Startpunkt, Zielpunkt und Routenverlauf sind durch das Konzept festgelegt worden. Durch diese gestalterischen Vorgaben war es einfach, die neu hinzukommenden Oberflächenelemente in das Gesamtbild einzufügen. Die Ballons sind bereits auf Auswahloptionen zur Start- und Zielfestlegung vorbereitet. Die Buttons müssen aber noch mit entsprechenden ClickHandlern belegt und auf die bereits beschriebene Art und Weise eingeblendet werden. Ebenfalls ist im Menü ein Navigationseintrag vorhanden, über welchen weitere Einstellungen vorgenommen werden können.

6.3.6 Layout

Was sich bei der Navigation als vorteilhaft erwiesen hat, soll auch für künftige Entwicklungen erhalten bleiben und fortgeführt werden. Aus diesem Grund wurde neben dem einheitlichen Farbkonzept aus Kapitel 5.3.3 auch eine einheitliche Icongestaltung festgelegt. Die zu verwendenden Icons stehen zur freien Nutzung für private und kommerzielle Projekte jeglicher Art zur Verfügung[3][4]. Sie sind als Vektorgrafiken vorhanden und können an das gültige Farbschema angepasst werden. Vorhandene Icons aus dem Projekt wurden gegen neue, passendere ausgetauscht. Damit wird der Gesamteindruck der App verbessert und das Design gefestigt.

6.4 PROBLEME

Während der Implementierung traten unterschiedlichste Probleme auf, welche an dieser Stelle erwähnt werden und nicht komplett im Rahmen dieser Arbeit behoben werden konnten. Ihnen sollte besondere Aufmerksamkeit gelten, wenn das Projekt in die nächste Entwicklungsphase geht.

6.4.1 Client-seitig

Auch in diesem Kapitel bietet sich die Einteilung in Client und Serverseite an. Nachfolgend werden die festgestellten client-seitigen Probleme erläutert:

Intents dienen dem Datenaustausch zwischen verschiedenen Programmkomponenten und Services. Sie sind Grundlage des Android-SDK's und somit Mittel der Wahl beim Aufrufen und Benachrichtigen von anderen Klassen mit Hilfe von Broadcasts. Leider wurden diese Aufrufe und Benachrichtigungen codeseitig kaum oder gar nicht dokumentiert. So fiel die Einarbeitung in den anfangs fremden Code sehr schwer und das Nachvollziehen von Abläufen war kaum möglich. Die Dokumentation wurde an wichtigen Stellen nachgebessert. Hundertprozentige Vollständigkeit kann aber auf Grund der begrenzten Zeit und der Projektgröße noch nicht gewährleistet werden.

Services laufen im Hintergrund und führen verschiedenste Datenverarbeitungen aus. **Activities** sind Bildschirmoberflächen, mit denen der Nutzer agiert. Beide können sich für bestimmte Systemereignisse registrieren und darauf reagieren. Sie müssen aber auch zu gegebenen Zeitpunkten beendet werden und sich als Eventempfänger wieder abmelden. Dieses Vorgehen war kaum oder fragwürdig umgesetzt und wurde zu Teilen nachgezogen. Um stabile Programmlaufzeiten zu erhalten, muss die korrekte Reaktion der App beim Starten, Beenden und Wechsel zwischen den Activities noch ausführlich getestet werden.

[3] `http://www.smashingmagazine.com/2011/12/29/freebie-free-vector-web-icons-91-icons` (Zugriff 29.12.2011)

[4] `http://www.augustinteractive.com` (Zugriff 24.02.2012)

Das LODOverlay ist die Klasse, welche in der aktuellen Version alle darzustellenden Geoobjekte enthält. Für jedes geladene Gebäude existiert ein Overlay auf der Karte. Dieses Konstrukt hat sich als nachteilig bei der Behandlung von Touchevents auf der Karte herausgestellt. Grundsätzlich sollte die Performance getestet werden, da alle Overlays auf Eingaben reagieren, obwohl sie nicht im Interaktionsradius des Nutzers liegen. Das heißt, dass unter Umständen Methodenaufrufe und Requests öfter als nötig ausgeführt werden. Das wiederum führt teilweise zu unabsehbaren Reaktionen und umständlichen Workarounds. Um die beste Implementierung zu finden, gilt es die Overlays zu evaluieren und ähnliche Probleme und deren Lösungsansätze ausfindig zu machen. Ein erster Ansatz wäre die Nutzung von ItemizedOverlays. Sie halten jeweils nur ein gezeichnetes Element und können auch gesondert via tap press angesprochen werden. Da die Raumdarstellung mitunter aber sehr viele zu zeichnende Objekte generiert, ist eine Evaluation diesbezüglich dringend notwendig.

6.4.2 Serverseitig

Die Probleme, die auf dem Server auftraten, sind auf die jeweiligen Serverinstanzen zurückzuführen und dementsprechend erklärt:

Der Directory Service (DS) ist mit einer Nutzeroberfläche ausgestattet, die noch nicht fehlerfrei arbeitet. Er kann in der jetzigen Version zum Ausspielen der Gebäuderserver-Urls genutzt werden, bietet aber keine fehlerfreie Möglichkeit diese zu bearbeiten. Es ist zu überlegen ob eine zentrale Geoserverinstanz die Aufgabe des DS übernehmen kann. Dies könnte ein Geoserver sein, welcher nur Url's und deren zugehörige Koordinaten verwaltet. Diese Arbeit hat gezeigt wie mächtig die Filterfunktionen des WFS sein können und wie sie umgesetzt werden.

Die Geoserverstabilität sollte Lasttests unterzogen werden. Beim Implementieren der verschiedenen Requests trat zweimal beim Geoserver ein schwerwiegender Fehler auf. Dieser führte zum Speicherüberlauf des Tomcatserver und legte so das gesamte System lahm. Die Fehler konnten nicht eindeutig reproduziert werden. Daher wäre ein gut konzipierter Stresstest für alle Serverkomponenten sinnvoll.

Die PostGIS-Datenbankeinrichtung scheint in der aktuellen Version noch Fehler aufzuweisen. Um solche grundlegenden Probleme in Zukunft zu vermeiden, muss ein Leitfaden zur Einrichtung erstellt werden und die aktuelle DB auf Fehleranfälligkeit und PostGIS-Kompatibilität überprüft werden.

6.5 ZUSAMMENFASSUNG

In diesem Kapitel wurden die Implementierungen und damit die Änderungen am aktuellen MapBiquitous-Projekt beschrieben. Um einen abschließenden Überblick zu geben und einen Vergleich zum Aus-

gangszustand herzustellen, werden die Änderungen kurz zusammengefasst. Es wird dabei auf die Konzeption verwiesen und eventuell abweichende Designentscheidungen werden erklärt.

Grundlage der meisten Änderungen war, dass neue Daten in Form von Points of Interest verwaltet werden. Die Datenhaltung ist im Konzept in der PostGIS-Datenbank vorgesehen (siehe 5.1.1). Dabei wurden noch keine Tabelleneigenschaften festgelegt. Die Entscheidung für die Tabellenstruktur, wie sie in Abschnitt 6.1.1 beschrieben wurde, ist aus Diskussionen mit den aktuellen Projektverantwortlichen und den Untersuchungen aus dem Kapitel „3 Verwandte Arbeiten" hervorgegangen. Dabei sind die Attribute der Tabelle jeder Zeit erweiterbar und nicht auf den implementierten Stand beschränkt, da der Client beliebig viele Attribute parsen und verwalten kann. Zu diesem Zweck wurde im Client das Modell *PointOfInterest* hinzugefügt, welches wie konzipiert vom *LocatableItem* erbt. Über dieses neue Modell lassen sich sowohl die vorhandenen LocatableItems abbilden, als auch die neuen POI's. Der gewünschte Vorteil tritt ein, da alle Suchergebnisse, auch die der externen Anbieter gleich behandelt und dargestellt werden können.

Um die Crowdsourcing-Idee umzusetzen und Daten mit Hilfe des Clients zu erstellen, wurde der Geoserver als neuer Gebäudeserver eingerichtet. Der in der Konzeption geplante Aufwand wurde deutlich überschritten (siehe 6.1.2). Die beschriebene Einrichtung mit Hilfe von Namensräumen und den weiteren Einstellungen, war notwendig, um die mögliche Skalierbarkeit im Framework einzuplanen. Gebäude und Etagen können dadurch getrennt verwaltet werden auch wenn sie sich (noch) auf dem physisch gleichen Server befinden. Als einzige Funktion, der in Abschnitt 5.2.3 konzipierten, wurde das Löschen von POI's noch nicht implementiert. Dieser Vorgang besitzt bis auf die gesendeten POST-Daten sehr viele Parallelen und kann daher innerhalb kurzer Zeit nachgezogen werden.

Das Konzept eines erweiterbaren Frameworks mit gekapselten Funktionalitäten, wie es in Kapitel 5.2 mit der Hilfe wichtiger Designpattern beschrieben wurde, konnte im Prototypen erfolgreich implementiert werden. Dazu mussten die bisherigen Strukturen aufgebrochen und neu sortiert werden. Das Ergebnis wird deutlich, wenn man die klare Struktur der Implementierung mit dem Ausgangszustand (beschrieben in Kapitel 2.4) vergleicht. Der Vorteil dieser Neuordnung ist, dass die entstandenen Komponenten unabhängig voneinander weiterentwickelt oder ausgetauscht werden können. Die Design-Pattern, wie zum Beispiel das MVC-Pattern wurden mit dem Android Design-Guide in Einklang gebracht und umgesetzt. Ein Beispiel für die erfolgreiche Umsetzung ist, das sämtliche Kommunikation in der neuen Implementierung aus dem loader-Package heraus passiert.

Das erarbeitete Oberflächendesign wurde ausführlich in Kapitel 5.3 beschrieben. Die zu nutzenden Touchscreen-Pattern wurden erläutert und deren Nutzung begründet. Bis auf wenige Feinheiten konnten sie wie in der Konzeption umgesetzt werden. Das Zoomen in der Karte funktionierte bereits durch die verwendete *MapActivity*. Diese Ansicht wurde aber vom Layout angepasst um das Erscheinungsbild abzurunden. Der double tap zum Vergrößern wurde nicht extra implementiert, da es sich um eine redundante Bedienungsart handelt und das zoomen bereist über Buttons, pinch und spread funktioniert. Es geht daher keine Funktionalität verloren. Für den Zugang zur Suche waren zwei Möglichkeiten im Konzept vorgesehen. Die eine über das Menü wurde genau so umgesetzt. Die andere über einen press wurde noch weiter optimiert. Statt eines press genügt nun ein simpler tap auf ein Such-Icon am unteren Kartenrand. Das macht das Öffnen

der Suche intuitiver, wenn der Nutzer die Funktion der „langen Bildschirmberührung“ nicht kennt. Weiterhin ermöglicht das Vorgehen auf konsistente Art und Weise weitere schnell erreichbare Funktionen am Kartenrand anzulegen. Der Menüpunkt der „Einstellungen“ ist ausgeblendet worden, da an dieser Stelle eine Evaluation fehlt, welche die richtige dem Nutzer zumutbare Menge an Einstellungen untersucht. Es ist durchaus auch denkbar die bestmöglichen Konfigurationen in weiteren Testreihen zu untersuchen und sie dann statisch vorzugeben. Die entwickelte Detailansicht wurde bisher nicht implementiert, da kein Bedarf zur Darstellung ausführlicher Daten bestand. Alle Informationen konnten in die Karte via Ballon eingebettet werden. Sollte sich der Datenhintergrund vergrößern und Bedarf zur Anzeige vieler Metadaten entwickeln, kann auf dieses Konzept ohne Weiteres zurück gegriffen werden. Ähnlich sieht es bei der Menüführung zur Navigation aus. Da diese Kernfunktionalität noch nicht vollständig entwickelt ist, gibt es auch noch keine streng nach Konzeption umgesetzte Menüführung.

Die Darstellung der Grundrisse und Räume wurde in einer zusätzlichen Befragung erarbeitet. Auf eine Beschriftung, wie sie ursprünglich zur ersten Orientierung gedacht war, wurde abgesehen, da die darunter liegende Karte Teilweise die Gebäudenamen bereits darstellt. Außerdem kann der Nutzer sehr schnell und effektiv durch einen tap auf den Grundriss den Gebäudenamen erfahren. Die aus der Umfrage resultierende Farbgebung (Kapitel 5.3.3), konnte direkt im Prototypen umgesetzt werden. Zusätzlich zum Konzept und zur Befragung, wurde ein schlichtes Iconset gewählt. Die unterschiedlichen vorhandenen Icons konnten im Sinne des Gesamtbildes ersetzt werden.

Zusammenfassend, ließ sich das erarbeitete Konzept sehr gut umsetzen. Einige Änderungen wurden gemacht, da sie sich als unpraktisch oder nicht notwendig erwiesen. Sehr wenige Funktionen wurden nicht implementiert, deren Umsetzung ist aber möglich und kann bei eintretendem Bedarf nachgeholt werden.

7 EVALUIERUNG

Vor dem Hintergrund der immer größer werdenden Verbreitung spielt die einfache Bedienung von Smartphones eine große Rolle. Die mehr als 800.000 weltweit neu aktivierten Geräte pro Tag[1] verdeutlichen die Wachstumsraten eindrucksvoll. Ein weiteres Kennzeichen der starken Verbreitung ist die riesige Zahl von teils konkurrierenden Apps, welche alle um die Nutzerschaft werben. Eine kostenpflichtige App kann im Android Market innerhalb von 15 Minuten zurück gegeben werden[2]. Der Nutzer hat also weniger als 15 Minuten Zeit, um sich für oder gegen eine Anwendung zu entscheiden. Ein unverständliches oder wider den Erwartungen reagierendes Userinterface kann dabei schnell zur Deinstallation und damit zur Rückgabe führen. An diesem Beispiel ist erkennbar, wie hoch der Druck auch aus wirtschaftlichen Perspektiven für den Entwickler ist, eine gute Software zu liefern.

Ausgereifte Usability und Userexperience sind daher aus vielerlei Hinsicht sinnvoll. Um die Usability zu überprüfen und mögliche Verbesserungspunkte aufzuzeigen, wird in diesem Kapitel eine Evaluierung durchgeführt, für welche zunächst die festgelegte Methode beschrieben wird. Zusätzlich werden in der ISO/IEC 9126 [fS01] Merkmale definiert, welche die Produktqualität einer Software sichern sollen. Die folgenden Qualitätsmerkmale (nach [fS01]) sind dabei von entscheidender Bedeutung:

Funktionalität legt fest, ob die Funktionen die geforderten Aufgaben erfüllen.

Zuverlässigkeit lässt sich daran erkennen, dass die Software ihr Leistungsniveau unter definierten Bedingungen über einen bestimmten Zeitraum aufrecht erhält.

Benutzbarkeit wird durch die Beurteilung einer Nutzergruppe festgelegt.

Effizienz zielt auf die verbrauchten Ressourcen in Vergleich zur erreichten Funktionalität.

[1] `http://www.nzz.ch/nachrichten/digital/850000_android-geraete_werden_taeglich_aktiviert_1.15337611.html` (Zugriff am 03.03.2012)

[2] `http://support.google.com/androidmarket/bin/answer.py?hl=de&answer=113515&topic=1046718&ctx=topic` (Zugriff am 03.03.2012)

Wartbarkeit beschreibt den Änderungsaufwand der Software, welcher für Korrekturen, Verbesserungen usw. notwendig wird.

Übertragbarkeit meint den Aufwand, der zu betreiben ist, um die Software in eine andere Umgebung zu übertragen.

Auch diese Merkmale sollen in der Evaluierung betrachtet werden und in Abwandlung auf das entstandene Framework angewendet werden.

7.1 UI-TEST

Wie in den einleitenden Worten dieses Kapitels erläutert, sind Usability-Tests mit Nutzern sehr wichtig, um die Funktionalität und Einsetzbarkeit zu überprüfen. Anschließend wird der durchgeführte Test kurz für das allgemeine Verständnis erklärt, danach werden die gewonnenen Ergebnisse ausgewertet und interpretiert.

7.1.1 Testaufbau

Das Zusammenarbeiten von Client und Server wurde in den bisherigen Kapiteln ausführlich beschrieben (siehe Kapitel 5 und 6). Es wurde auch für die Nutzertests nicht verändert. Das heißt, dass Directory Service, Geoserver und letztlich die PostGIS-Datenbank die Anwendungsdaten bereitstellen und aufarbeiten. Das Client-Gerät war bei allen Tests das Galaxy Nexus von Samsung. Als Betriebssystem dient Android 4.0.2 mit der Kernel-Version 3.0.8-gaaa2611. Internetzugang erhielten die Testpersonen über WLAN.

7.1.2 Testmethoden

Um die in der Einführung genannten Qualitätsmerkmale anhand einer Nutzerbefragung zu überprüfen, wurde ein Test aus verschiedenen Methoden entwickelt.

Die erste Methode stellt das **Task Based Testing**[3] dar. Dabei werden den Testpersonen typische Aufgaben für den Umgang mit der Software gestellt. Um diese zu lösen, sollten möglichst alle zu überprüfenden Funktionen der App in den Aufgaben enthalten sein. Dieses Vorgehen ermöglicht dem Nutzer die App ausführlich zu testen und sich ein breites Bild über die Funktionalität bilden zu können. Ähnliche Tests, die das Verhalten des Nutzers bei der Erfüllung typischer Aufgaben auswerten, sind der **Out-of-the-Box-Test** oder die **Feldbeobachtung** (beide [HPSG11]).

Die Tests ermöglichen eine umfassende Bewertung des implementierten Prototypen. Die Anzahl der Testpersonen beläuft sich bei den benannten Test zwischen fünf und 15 Probanden. Dass das Testen mit wenigen Personen im Usability-Bereich sehr effizient ist, bestätigt die Untersuchung

[3] `http://en.wikipedia.org/wiki/Usability_testing` (Zugriff am 24.02.2012)

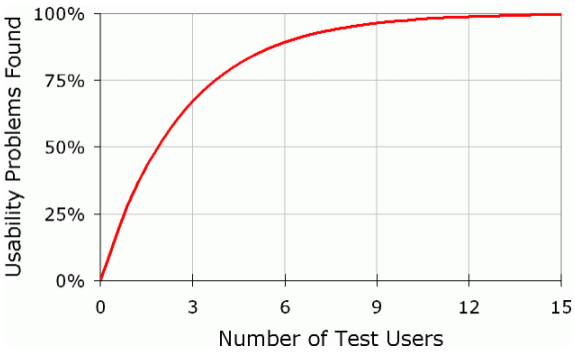


Abbildung 7.1: gefundene Usability-Probleme in Bezug auf Anzahl der Testpersonen (siehe [Nie00])

von [Nie00]. Das Verhältnis zwischen der Anzahl der Tester und den gefundenen Problemen stellt Nielsen als Ergebnis seiner Untersuchung in Abbildung 7.1 dar. Man kann erkennen, dass mit fünf Personen bereits 80% aller Usability-Probleme entdeckt werden. Bei jeder weiteren Testperson sinkt die Anzahl der neu gefundenen Probleme, das Testen wird somit aufwendiger und uneffektiver.

Die vollständigen Aufgaben und Fragen des UI-Tests befinden sich im Anhang A.3. Die erste Aufgabe zielt darauf ab, die Symbole bzw. Grundrisse der Gebäude zu erkennen und durch ihren Namen zu identifizieren.

> Wie viele und welche Gebäude können Sie auf dem Campus der TUD erkennen?

Mit der zweiten Aufgabe soll der Nutzer Räume identifizieren lernen und Etagen auswählen.

> Wie viele Räume hat der Barkhausenbau („tud_bar:TUD_BAR“) in der 5. Etage?

Die dritte Aufgabe beinhaltet die Nutzung der Suche und das Auswählen von Räumen um Informationen zu erhalten.

> Suchen Sie in der Fakultät Informatik („tud_inf:TUD_INF“) Raum „E023“ und schreiben Sie die angegebene Nutzung des Raumes auf!

Aufgabe Nummer vier bringt dem Nutzer die aktuell implementierte Navigationsfunktion näher. Der Tester durchläuft dabei den kompletten Navigationsprozess.

> Wählen Sie den Raum „E023“ als Start für die Navigation aus. Wählen Sie danach die Alte Mensa („tud_m13:TUD_M13“) als Ziel aus und starten Sie die Navigation! Lassen Sie sich Schritt für Schritt zum Ziel navigieren.

Die letzte Aufgabe komplettiert den Funktionsumfang mit dem Einfügen eines Points of Interest.

Fügen Sie durch einen „longClick“ (langes Berühren einer Stelle der Karte) einen erdachten „Point of Interest“ in die Karte ein.

Die Tests erfordern in ihrer Vollständigkeit die Beobachtung (nach [BN06]), sowie Protokollierung der Nutzeraktivitäten. Die sogenannte **protokollierte Nutzung** ermöglicht es, Verhalten und Fehlverhalten der Testperson nach einem einheitlichen Protokoll aufzuzeichnen. Die Vereinheitlichung macht das Vergleichen der Ergebnisse möglich. Es wurden folgende Fakten bei jedem Probanden festgehalten:

- Ob und welche Hilfestellungen bei der Nutzung gegeben werden mussten.
- Wie lange sich der Tester mit der App vertraut machte und wie hoch die gesamte Testzeit war.
- Für jede Aufgabe wurden Eingabefehler und die Effizienz der Ausführung bewertet.
- Sonstige Bemerkungen sowie Logeinträge bei Programmabstürzen wurden ebenfalls aufgezeichnet.

Als Bewertungsgrundlage der Usability wird die **System Usability Scale** (SUS) angewendet. Bei SUS (nach [Bro96]) handelt es sich um eine einfache zehn Punkte Likert-Scale (siehe [Lik32]) mit deren Hilfe sich eine aussagekräftige Kennzahl ableiten lässt. Diese Kennzahl errechnet sich aus den Werten der Skala und befindet sich im Wertebereich zwischen 0 und 100. Laut [Bro96] kann mit dieser Wertung auch die Benutzbarkeit verschiedener Softwareprodukte verglichen werden. Sie dient ebenfalls hervorragend zur begleitenden Untersuchung der Entwicklung einer Nutzeroberfläche. Zu diesem Zweck sollte dieselbe Befragung mit den nächsten Entwicklungsstufen der App erneut ausgeführt werden. Die Verbesserung oder Verschlechterung der Usability kann so über einen größeren Zeitraum festgestellt werden.

Die Aussagen des Originaltests wurden minimal modifiziert und übersetzt, um den Anforderungen der Testkonfiguration zu genügen:

1. Ich würde die App öfter nutzen.
2. Ich fand die App unnötig komplex.
3. Es war einfach die Anwendung zu nutzen.
4. Ich benötige Hilfe um die App richtig zu nutzen.
5. Ich fand die verschiedenen Funktionen des Systems gut in die Anwendung integriert.
6. Es gibt zu viele Inkonsistenzen in der App.
7. Ich denke, dass die meisten Personen die Nutzung des Programms schnell erlernen.
8. Die Anwendung war sehr umständlich zu verwenden.
9. Beim Nutzen der App habe ich mich sicher gefühlt.

10. Ich musste viel lernen um die App nutzen zu können.

Die Berechnung der Usability-Kennzahl wird im Auswertungskapitel 7.1.3 beschrieben.

7.1.3 Auswertung

Die befragten Personen waren alle zwischen 22 und 39 Jahren alt. Sie wiesen durchweg gute Kenntnisse im Umgang mit Kartendiensten im Internet auf und konnten mindestens zwei verschiedene Anbieter nennen. Bei der Wahl der Testpersonen wurde darauf geachtet, dass sowohl Smartphone-Besitzer, als auch in diesem Bereich unerfahrene Nutzer, angesprochen wurden. In der Gesamtheit ergab sich eine Verteilung von 60% zu 40% zwischen Smartphonebesitzern und unerfahrenen Nutzern. Von den erfahrenen Nutzern sind alle an die Verwendung von Karten-Apps gewöhnt. Die Hälfte der Probanden hatte bereits etwas über das MapBiquitous-Projekt gehört. Es wurde trotzdem allen Teilnehmern eine kurze Einführung über Umfang und Ziele des Projekts gegeben.

Die Aufgaben, welche zum Kennenlernen der Anwendung dienten, wurden von allen Teilnehmern korrekt gelöst. Viele der Personen wünschten sich eine Erklärung und kurze Hinweise zu den einzelnen Funktionen. Beim Testen und Erfüllen der Aufgaben, ließen sich alle Teilnehmer mehr als 25 Minuten Zeit. Sie waren bemüht alle Funktionalitäten zu verstehen und generierten eigene Testfälle.

SUS Frage	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
1	4	4	4	4	5	4	5	2	4	4
2	2	1	1	1	3	1	2	3	3	2
3	4	4	4	4	4	5	4	3	3	4
4	3	4	3	2	4	1	4	5	3	3
5	4	4	4	5	5	4	4	3	3	5
6	2	2	2	1	1	1	1	4	2	1
7	4	4	4	4	5	3	5	3	4	5
8	2	2	2	2	2	2	1	3	3	2
9	4	3	4	2	3	3	4	3	4	4
10	2	3	2	1	4	1	2	4	3	1
SUS	72,5	67,5	75	80	70	82,5	80	37,5	60	82,5

Tabelle 7.1: Ergebnisse der SUS-Fragen

7.1.4 Auswertung der System Usability Scale

Pro Testperson wird aus den zehn beantworteten SUS-Fragen ein Wert berechnet, der zwischen 0 und 100 liegt. Dieser Wert berechnet sich wie folgt: Alle angekreuzten Werte (immer zwischen 1 und 5) der ungeraden Fragen werden mit 1 subtrahiert. Die Werte der anderen Fragen werden

von 5 abgezogen. Die zehn Ergebnisse werden aufsummiert und mit 2,5 multipliziert. Nach Sauro [Sau11] liegt die durchschnittliche SUS-Kennzahl bei 68. Die Auswertung in Tabelle 7.1 belegt, dass der Score für MapBiquitous mit 70,75 Punkten leicht über dem Durchschnitt liegt. Durch die Untersuchungen von [BKM08] kann dieser SUS-Kennzahl das beschreibende Attribut „gut (bedienbar)“ zugeordnet werden. Diese Einordnung zeigt jedoch auch, dass es durchaus verbesserungswürdige Bereiche gibt.

Die durchschnittlich beste Bewertung erhielt die Frage sechs, gefolgt von den Fragen zwei, fünf und sieben. Damit wurden Fragen, die die Konsistenz und die Integration der Funktionalität in das Gesamtbild bewerten, als sehr gut beurteilt. Eine neutrale, bzw. sehr leicht ablehnende Bewertung erhielten die Fragen neun und vier. Bei diesen Fragen ging es um die notwendige Hilfestellung, die während der Nutzung gegeben werden musste. Das heißt, dass die Testpersonen Hilfe in Anspruch genommen haben, um die Funktionen zu erlernen. Dieser Punkt zeigt eine der deutlich gewordenen Schwächen: Die intuitive Nutzung kann und muss noch verbessert werden. Vorstellbar wären kleine Tipps, die beim ersten Gebrauch der App eine Hilfestellung geben. Üblicher Weise wird der Nutzer kein Handbuch lesen und will die Funktionen durch Ausprobieren lernen. Das sollte in der weiteren UI-Entwicklung beachtet werden.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Durchschnitt
Übersichtlichkeit	4	5	4	5	5	5	5	4	4	5	4,6
Design	5	4	4	2	4	4	5	4	4	5	4,1
Verständlichkeit	5	3	3	4	4	4	4	3	3	4	3,7
Navigation	5	2	2	4	5	5	1	2	3	4	3,3
Beschriftung	5	3	4	3	4	3	4	2	2	4	3,4
Menüführung	3	4	3	4	5	4	3	2	3	4	3,5

Tabelle 7.2: Ergebnisse der allgemeinen Fragen

7.1.5 Bewertung der einzelnen Bestandteile

Die Verteilung der Bewertung auf die einzelnen Bestandteile der Anwendung wurde mit sechs weiteren Kategorien abgefragt. Dabei kreuzten die Testpersonen auf einer Skala von eins (nicht gefallen) bis 5 (gefallen) ihre Wertung an. Diese allgemeine Einschätzung zu Layout und Funktionsumfang ist in Tabelle 7.2 dargestellt. Wie bereits bei der SUS-Auswertung werden Übersichtlichkeit und Design als sehr gut eingestuft. Abzüge gab es bei der Bewertung der Navigationsfunktion. Die Beobachtungen der Testpersonen bestätigen dieses Ergebnis. So mussten bei der vierten Aufgabe die meisten Hilfestellungen gegeben werden. Die zum Ziel führenden Interaktionen waren den Nutzern nicht intuitiv klar und mussten erklärt werden. Dabei ist zu bedenken, dass die Navigationsfunktionalität sich noch in der Entwicklung befindet. Die endgültige Eingliederung in das Userinterface steht also noch aus.

7.1.6 Kritikpunkte und Verbesserungsvorschläge

Durch die Protokollierung sind vor allem Kleinigkeiten aufgefallen, die bei der Nutzung eine optimale Bedienung verhinderten. Auch die Testpersonen gaben an den vorgesehenen Stellen Feedback zu Verbesserungsmöglichkeiten. Die Vorschläge werden hier kurz erläutert:

Bei der aktuell „simulierten" Position des Nutzers, als auch bei der zukünftig korrekten Positionierung, sollte an passenden Stellen das Gebäude vorausgewählt sein. Dies betrifft zum Beispiel die Suche oder Navigation.

Die Wegbeschreibungen der Outdoor-Navigation zeigen noch HTML-Tags an. Dieses Verhalten ist auf geänderte Parserfunktionen unter Android 4 zurückzuführen und muss behoben werden. Das bereits angesprochene komplizierte Navigationsmenü wurde auch an dieser Stelle von den Nutzern zur Verbesserung aufgeführt. Als Extra-Feature könnte die Zeit bis zum Erreichen des Ziels angezeigt werden.

Der press auf die Karte sollte durch ein haptisches Feedback deutlich gemacht werden. In der Diskussion stand auch, dass der erscheinende Ballon nicht erst beim loslassen, sondern nach der abgelaufenen Mindestzeit des press angezeigt werden könnte.

Bei kleineren Zoomstufen wurde eine bessere Abgrenzung der einzelnen Räume gewünscht. Wenige Räume in einer Etage verhindern zudem den Etagenwechsel. Die Auswahl einer anderen Etage sollte möglich sein, auch wenn auf einen Teil des Grundrisses ohne eingezeichneten Raum geklickt wird.

Es fiel auf, dass die Liste bei Raum- oder Etagenwahl nicht immer einheitlich sortiert war. Als zusätzliches Feature könnte an dieser Stelle ein Autocomplete-Feld oder ein Filter während der Eingabe die Nutzbarkeit verbessern.

Nach dem Erstellen eines Points of Interest sollte dieser direkt angezeigt werden. Andere POI's die durch die Suche gefunden wurden, benötigen noch die Funktion zum Anzeigen ihrer Details. Zur Verbesserung der Suche wurde zusätzlich der Vorschlag eingebracht, direkt auf die gefundenen Ergebnisse zu zoomen.

7.2 FRAMEWORK

Die Implementierung der neuen Location-Based Services ist bereits als Evaluierung der Praxistauglichkeit des Frameworks anzusehen. Im entstandenen lauffähigen Prototypen konnten die Konzepte umgesetzt und getestet werden. Dieser Umstand spricht für eine gute Konzeption und Planung. Dass bei der Softwareentwicklung Hindernisse auftreten (siehe Kapitel 6.4), ist dem fortlaufenden Entwicklungsprozess geschuldet und damit völlig normal. Alle geschilderten Schwierigkeiten sind lösbare Aufgaben, die in und mit dem Framework behoben werden können. Die teilweise angegebenen Lösungsvorschläge belegen dies deutlich.

Als weitere positive Entwicklung ist die zunehmende Codedokumentation zu werten. Sie erleichtert dem Entwickler die Arbeit und den Einstieg in das Framework. Weitere Entwickler sollten die strikte Einhaltung des **JavaDoc**-Styles beibehalten. Die JavaDoc's wurden in Eclipse via „Ant Build" generiert und dem Projekt beigefügt. In der dazu notwendigen javadoc.xml sind alle zu dokumentierenden Packages und classpath's angegeben. Die ausgeführte Konfiguration ist im Anhang A.5 beigefügt. Projektspezifische Pfade müssen natürlich ersetzt werden. Die Dokumentation des Projektes ist damit vollständig aufgearbeitet und an Entwicklungsstandards angepasst.

Da das Framework prototypisch in Android umgesetzt wurde, stehen mit den *Android Developer Tools* weitere Validierungsmechanismen zur Verfügung. Ein hilfreiches Tool ist dabei **Android Lint** (seit ADT 16[4]). Lint überprüft das Projekt auf Fehler und gibt dem Entwickler Hinweise zu deren Lösung. Es werden unter anderem Fehler im Manifest, Layoutprobleme, ungenutzte Ressourcen und eventuelle Usability-Probleme aufgedeckt. Bei der Überprüfung von MapBiquitous wurden keine schwerwiegenden Fehler entdeckt.

7.3 FAZIT

Zusammenfassend soll am Ende der Evaluierung ein Fazit gezogen werden. Dazu werden die anfänglich in Kapitel 4 entwickelten Anforderungen aufgegriffen.

Die Gruppe der Anforderungen A1 und A2 wurden durch die in Abschnitt 7.2 beschriebene Evaluierung überprüft. Die festgelegten Anforderungen zur Architektur und technischen Umsetzung wurden komplett während der Konzeption (A1.1) und Implementierung erfüllt. Sie waren die Grundvoraussetzung für einen lauffähigen Prototypen (A1.2). Die Datenverwaltung auf der Serverseite (A1.3, A1.4) wurde mit Hilfe des Geoservers bewältigt. Die dadurch hinzukommenden WFS-Funktionen konnten auf dem Client umgesetzt und getestet werden (A1.5 bis A1.6). Dazu gehört auch das Implementieren der neuen Package-Struktur, der erweiterten Konfigurationsmöglichkeiten (A1.8) und der Userinterface-Kapselung (A1.7). Durch die beschriebenen Frameworktests wurde eine fehlerfreie Arbeitsweise nachgewiesen. Die Anforderungen ab A2.1 definieren den Datenaustausch zwischen dem Server und dem Client. Sie wurden client-seitig implementiert und letztlich durch die Framework- und Nutzerevaluierung getestet. Es konnten als externe Dienstanbieter (A2.2) Google Places und OpenStreetMap integriert werden. Das Suchen und Einfügen von Daten, wie es in den Anforderungen A2.3 bis A2.7 bzw. A2.9 beschrieben ist, wurde umgesetzt. Einzige Einschränkung ist die bisher nur case-sensitive Suche auf Gebäudeserverdaten. An der Navigation (A2.8) wird im Rahmen einer weiteren Diplomarbeit gearbeitet. Die Implementierung ist daher noch nicht abgeschlossen.

Die Umsetzung der Anforderungen unter A3 konnte durch die Nutzerevaluation in Abschnitt 7.1 belegt werden. Die Ausnahme bildet hier A3.5 (Detailansicht eines Suchergebnisses). Diese Darstellung wurde noch nicht implementiert, da die bisher in den Datengrundlagen vorhandenen Informationen effektiver direkt in der Karte angezeigt werden konnten (Infoballon, siehe 6.3.2). Die Anforderung ist dadurch nicht hinfällig, da zukünftige Daten, zum Beispiel Fotos, durchaus eine gesonderte Darstellung verlangen. Bei der Evaluation ergaben sich für die Anforderungen

[4] `http://tools.android.com/tips/lint` (Zugriff am 05.03.2012)

A3.6 und A3.7 (UI und UX) neue Features, die in den nächsten Projektversionen implementiert oder berichtigt werden sollten. Es handelt sich dabei um notwendige Verbesserungen für eine ausgeglichenere Usability. Die Personalisierungsoptionen unter A3.8 wurden bisher nur auf Basis statischer Konfigurationsmöglichkeiten implementiert. An dieser Stelle sollte gesondert entschieden werden, welche dynamischen Anpassungen der Nutzer in der App selber vornehmen können sollte.

Am Ende des Kapitels 4: Anforderungen wurden zwei verschiedene Zielgruppen in A4 festgelegt. Die Entwickler konnten durch die Evaluation des Frameworks bereits berücksichtigt werden. Das heißt, es wurde gezeigt, dass Anforderung A4.1 durch eine strukturierte Dokumentation des Frameworks erfüllt ist. Die Anwender spielten die entscheidende Rolle in den UI-Tests und lieferten in diesem Rahmen wertvolle Ergebnisse. Die Anforderung A4.2 nach Usability-Tests ist daher auch erfüllt.

8 ZUSAMMENFASSUNG UND AUSBLICK

8.1 ZUSAMMENFASSUNG

Zu Beginn dieser Arbeit wurden Location-Based Services vorgestellt und deren aktuelle Verbreitung beschrieben. Die große Bedeutung für die breite Nutzerschaft sowie das Potential, welches noch in der Entwicklung steckt, wurde deutlich. Die Motivation für die Entwicklung eines integrierten Indoor/Outdoor-Location-Based Service Framework war daher schnell gefunden. Die Grundlagen, die in Kapitel 2 vermittelt wurden, sollten helfen, die aktuellen Standards zu verstehen. Es wurden wichtige Begriffe und Funktionalitäten erläutert, auf die im Verlauf der Arbeit zurück gegriffen werden konnte. Zusätzlich wurde ein Einblick in den aktuellen Status des MapBiquitous-Projektes gegeben. Damit ließ sich die notwendige Entwicklung einschätzen und letztlich die Anforderungen besser abgrenzen.

Das Kapitel 3, „Verwandte Arbeiten", stellte die wichtigsten Projekte und deren Zugang zu Location-Based Services vor. Es wurden Funktionalitäten analysiert und verglichen. Dokumentationen und vorhandene Implementierungen wurden recherchiert und auf ihre Tauglichkeit für den Indoor-Bereich überprüft. Trotz der Vielzahl von Ansätzen und Möglichkeiten konnte gezeigt werden, dass noch keine ausgereiften Indoor-LBS existieren. Google Maps hat als Marktführer die weitesten Fortschritte gemacht, aber auch hier waren noch keine „Indoor-Services" in dem Sinne für die Nutzung verfügbar. Dieses Kapitel war die Voraussetzung für die Entwicklung von sinnvollen Anforderungen im Kapitel 4. Zu den Teilgebieten *Architektur*, *Schnittstellen* und *Userinterface* konnten die erforderlichen Funktionen abgegrenzt werden. Im folgenden Kapitel wurde die Dreiteilung der Anforderungen aufgegriffen und ein Konzept für das gewünschte Framework entwickelt. Alle gewonnenen Erkenntnisse aus vorangegangenen Kapiteln wurden angewandt, um ein Framework zu konzipieren, welches aktuellen und zukünftigen Gegebenheiten gewachsen sein wird. Die Projektstruktur wurde komplett überarbeitet und die Nutzeroberfläche in weiten Teilen angepasst. Bei der Konzeption wurde auf Softwaredesign und populäre Entwicklungsprin-

zipien geachtet. Es wurden bekannte Design-Pattern und neue Interaktionsmuster miteinander verflochten und zu einer Anwendung verschmolzen. Für das MapBiquitous-Projekt wurde im Rahmen der Arbeit ein flexibles, wiederverwendbares Framework definiert. Dieses Grundgerüst ermöglicht erstmals die Kapselung der einzelnen Funktionen. Es macht das Projekt für zusätzliche Komponenten ausbaufähig und für die Entwicklung übersichtlicher.

Die Implementierung (Kapitel 6) stellte im nächsten Abschnitt den sogenannten „Proof of Concept" - die Überprüfung in der praktischen Umsetzung - dar. Die meisten der konzeptionell ausgearbeiteten Ideen wurden mit Hilfe des MapBiquitous-Prototypen umgesetzt und getestet. Schwierigkeiten wurden aufgedeckt und Lösungsansätze ausprobiert. Dadurch wurden dem Projekt während der Implementierungsphase die wichtigsten Location-Based Services hinzugefügt. Die gewünschte Integration von Indoor und Outdoor-Services wurde erreicht. Um diese auch in der Nutzeroberfläche widerzuspiegeln, wurden Interaktionselemente neu angeordnet und eingefügt.

Die folgende Evaluierung bestätigte die Konzeption in ihrer umgesetzten Art und Weise. Die Untersuchung umfasste einerseits Nutzertests, welche die Usability der Anwendung bewerteten und andererseits die Beurteilung des implementierten Codes und dessen Dokumentation als Teil des Frameworks. Die UI-Tests können weiterhin als Vergleichswert für zukünftige Entwicklungsversionen dienen. Die erfolgreichen Strategien zur Dokumentation und zur Erweiterung des Frameworks sollten ebenfalls in den nächsten Versionen des Projektes Anwendung finden, um kontinuierlich gute Entwicklungsarbeit leisten zu können.

8.2 AUSBLICK

Da die Entwicklung eines Frameworks nie wirklich abgeschlossen ist, sollen in diesem Kapitel der Arbeit Ideen zur Erweiterung vorgestellt werden. Es gibt viele Möglichkeiten, das Framework auszubauen und zu optimieren. Dabei sind Optimierungen eher ein Gewinn für die Stabilität des Systems, wie zum Beispiel das Schreiben von Tests. Unit-Tests oder andere Methoden der Funktionsprüfung, sind ein fester Bestandteil jeder Softwareentwicklung. Der fehlenden Zeit und der Komplexität des Testthemas ist es geschuldet, dass im Rahmen dieser Arbeit noch keine Entwicklungen in dieser Richtung stattgefunden haben. Es würde sich in jedem Fall als Gewinn für das Framework erweisen, wenn in regelmäßigen Abständen und nach dem Implementieren neuer Funktionen Standardtests ausgeführt werden könnten. Diese würden den gleichbleibend hohen Qualitätsanspruch zusätzlich sichern. Auch Fehler in neuen Funktionen könnten dadurch schnell und effektiv entdeckt werden.

In der Zeit immer größer werdenden Datenaustauschs ist auch die Datensicherheit ein zu beleuchtender Punkt. Durch diese Arbeit wurden dem Prototyp bereits Möglichkeiten zum Crowdsourcing gegeben. Viele Nutzer können den Datenbestand kontinuierlich vergrößern und aktualisieren. Diese Möglichkeiten müssen vor Missbrauch geschützt werden. Die Ansätze um Zugänge zu schützen und Datenströme zu verschlüsseln sind vielfältig. Zugriffsrechte auf bestimmte Funktionen sind denkbar und auch eine damit verbundene Nutzerverwaltung könnte dem MapBiquitous-Projekt einen Mehrwert geben. Zu den Sicherheitsaspekten gehört aber auch die Möglichkeit,

Daten zu sichern und Backups durchzuführen.

Ein weiterer Aspekt ist die Portierung des Prototypen oder die Implementierung des Frameworks für andere Plattformen. Eine der Anforderungen war es, die Konzeption plattformunabhängig durchzuführen, um sich nicht auf eine Art von mobilen Endgeräten festzulegen. Nun ist es an der Zeit, Lösungen zu finden, welche die App auf IOS und Windows Geräten lauffähig macht oder eine webbasierte Version des Frameworks zu entwickeln. Vorhandene LBS sollten dabei natürlich erhalten bleiben.

Zum Abschluss des Kapitels steht noch ein Blick für zukünftige Entwicklungen aus. *Augmented Reality* und 3D-Technologien sind Schlagworte, die immer wieder in der aktuellen Softwareentwicklung fallen. Google Street View stellt reale Aufnahmen als Location-Based Service dar und vermittelt dadurch das Gefühl von Dreidimensionalität. Ähnlich werden durch Augmented Reality bereits jetzt Zusatzinformationen auf besondere Gebäude „projiziert". Leistungsstärkere mobile Geräte und neue Sensoren ermöglichen einen zusätzlichen Funktionsumfang. Die Erforschung dieser Möglichkeiten im Indoor-Bereich bietet vielfältige Aufgabenstellungen für weitere Forschungsarbeiten.

A EVALUATION

A.1 FRAGEBOGEN AB TEST

1. Frage / question:

Sind die Gebäude deutlich erkennbar?
Are the buildings clearly visible?

ja / yes	nein / no

Welche Darstellung hebt die Grundrisse besser hervor?
Which representation points out the floor plans in a better way?

Bild / picture 1	Bild / picture 2

2. Frage / question:

Beurteilen Sie die Aussage: Karte und Grundrisse wirken harmonisch zueinander!
Evaluate the statement: Map and floor plans are harmonious with each other!

	trifft zu / strongly agree	trifft eher zu / agree	trifft eher nicht zu / disagree	trifft nicht zu / strongly disagree
Bild / picture 1				
Bild / picture 2				
Bild / picture 3				
Bild / picture 4				
Bild / picture 5				
Bild / picture 6				

3. Frage / question:

Beurteilen Sie die Aussage: Sonderräume sind gut erkennbar!
Evaluate the statement: Special rooms are easily recognizable!

	trifft zu / strongly agree	trifft eher zu / agree	trifft eher nicht zu / disagree	trifft nicht zu / strongly disagree
Bild / picture 1				
Bild / picture 2				
Bild / picture 3				
Bild / picture 4				

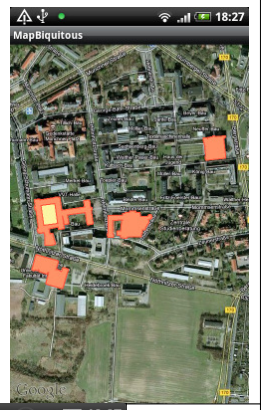
4. Frage / question:

Beurteilen Sie die Aussage: Der ausgewählte Raum ist deutlich von den anderen zu unterscheiden!
Evaluate the statement: The selected room is clearly distinguishable from the other!

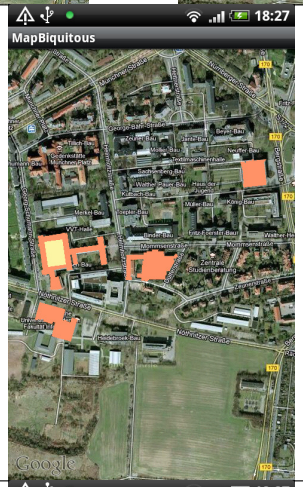
	trifft zu / strongly agree	trifft eher zu / agree	trifft eher nicht zu / disagree	trifft nicht zu / strongly disagree
Bild / picture 1				
Bild / picture 2				

A.2 VERGLEICHSFOLIEN DES AB TESTS

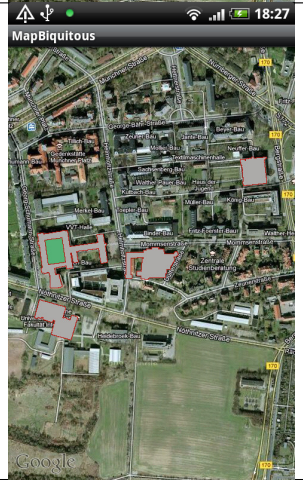
1. Frage / question



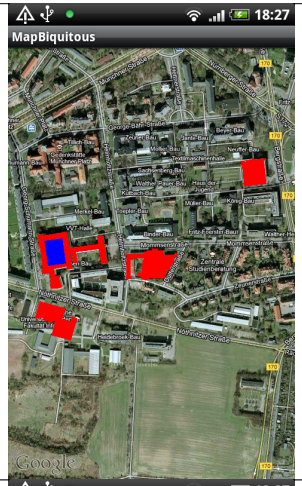
2. Frage / question



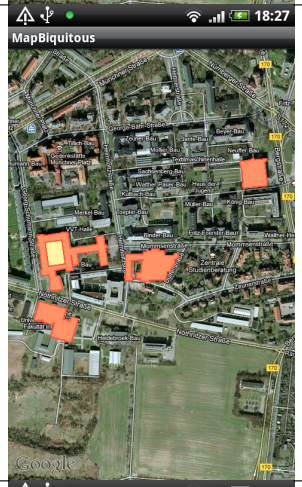
2. Frage / question



2. Frage / question

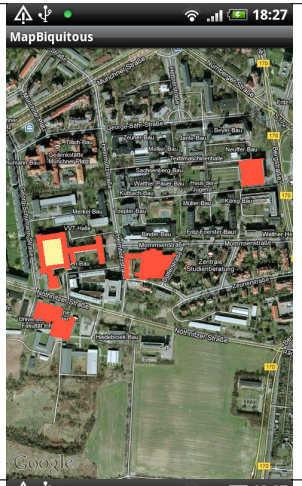


2. Frage / question



3. Frage / question

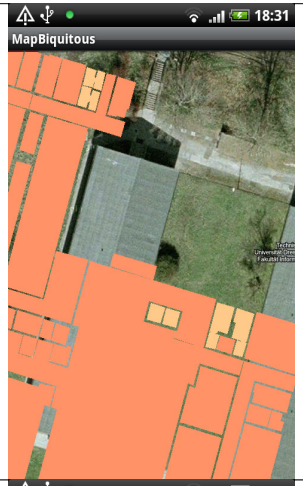
2. Frage / question



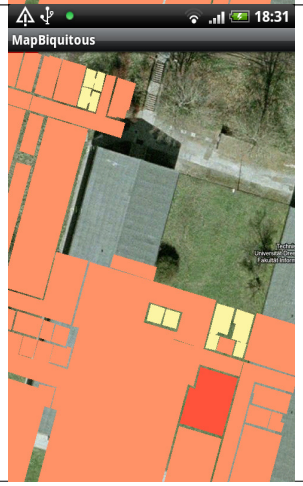
2. Frage / question

3. Frage / question

3. Frage / question



4. Frage / question



3. Frage / question

4. Frage / question

A.3 AUFGABEN UND FRAGEN DER ABSCHLUSSEVALUATION

Evaluation des Mapbiquitous Userinterface

Alter: _____

Geschlecht: ☐ männlich ☐ weiblich

Beruf: _____

Kennen Sie Kartendienste aus dem Internet? Wenn ja welche?

_____

_____

Besitzen Sie ein Smartphone oder haben Sie ein Smartphone schon intensiver genutzt? ☐ ja ☐ nein

Wenn ja, haben Sie Kartendienste bereits auf einem Smartphone genutzt? ☐ ja ☐ nein

Kennen Sie das MapBiquitous-Projekt der TU-Dresden? ☐ ja ☐ nein

Wenn ja, haben Sie den Prototypen dieses Projekts bereits bedient? ☐ ja ☐ nein

Einführung:

Diese Umfrage dient der Untersuchung des MapBiquitous-Prototypen. Die Android-App stellt verschiedene Gebäude der TU Dresden auf einer Karte dar. Nehmen Sie sich ein paar Minuten Zeit um sich mit der Anwendung vertraut zu machen und lösen Sie dann die folgenden Aufgaben. Wenn Sie Probleme haben oder in der Lösung nicht weiter kommen, fragen Sie bitte nach.

Hinweis zur Smartphonenutzung: Die Tastatur blenden Sie mit den „Zurück-Button“ aus. Das Menü der App erreichen Sie über:

Aufgaben:

1. *Wie viele und welche Gebäude können Sie auf dem Campus der TUD erkennen?*

_____

2. *Wie viele Räume hat der Barkhausenbau („tud_bar:TUD_BAR“) in der 5. Etage?*

_____

3. *Suchen Sie in der Fakultät Informatik („tud_inf:TUD_INF“) Raum „E023“ und schreiben Sie die angegebene Nutzung des Raumes auf!*

_____

4. *Wählen Sie den Raum „E023“ als Start für die Navigation aus. Wählen Sie danach die Alte Mensa („tud_m13:TUD_M13“) als Ziel aus und starten Sie die Navigation! Lassen Sie sich Schritt für Schritt zum Ziel navigieren.*

_____

5. *Fügen Sie durch einen „longClick“ (langes Berühren einer Stelle der Karte) einen erdachten „Point of Interest“ in die Karte ein.*

_____

Bewerten Sie mit den gewonnen Erkenntnissen die folgenden Aussagen
(1 = ich stimme gar nicht zu, 5 = ich stimme voll zu):

	1	2	3	4	5
Ich würde die App öfter nutzen.	☐	☐	☐	☐	☐
Ich fand die App unnötig komplex.	☐	☐	☐	☐	☐
Es war einfach die Anwendung zu nutzen.	☐	☐	☐	☐	☐
Ich benötige Hilfe um die App richtig zu nutzen.	☐	☐	☐	☐	☐
Ich fand die verschiedenen Funktionen des Systems gut in die Anwendung integriert.	☐	☐	☐	☐	☐
Es gibt zu viele Inkonsistenzen in der App.	☐	☐	☐	☐	☐
Ich denke, dass die meisten Personen die Nutzung des Programms schnell erlernen.	☐	☐	☐	☐	☐
Die Anwendung war sehr umständlich zu verwenden.	☐	☐	☐	☐	☐
Beim Nutzen der App habe ich mich sicher gefühlt.	☐	☐	☐	☐	☐
Ich musste viel lernen um die App nutzen zu können.	☐	☐	☐	☐	☐

Welche Elemente der Seite haben Ihnen besonders gefallen?
(1 = nicht gefallen, 3 = keine Meinung, 5 = gefallen)

Übersichtlichkeit	☐	☐	☐	☐	☐
Design	☐	☐	☐	☐	☐
Verständlichkeit	☐	☐	☐	☐	☐
Navigation	☐	☐	☐	☐	☐
Beschriftung	☐	☐	☐	☐	☐
Menüführung	☐	☐	☐	☐	☐

Haben Sie bei der Nutzung Informationen vermisst? ☐ ja ☐ nein

Wenn ja, welche?

_____

_____

Lief die Anwendung stabil und ohne Laufzeitfehler? ☐ ja ☐ nein

Wenn Fehler auftraten, welche?

_____

_____

Haben Sie weitere Verbesserungsvorschläge für die Anwendung?

_____

_____

_____

_____

A.4 PROTOKOLL DER ABSCHLUSSEVALUATION

Protokoll für die Beobachtung

Mussten Hilfestellungen gegeben werden? ☐ ja ☐ nein

Wenn ja, welche?

_____

_____

Wie schnell kam der Nutzer mit dem Interface zurecht?

Minuten: _____

Wie lange dauerte die gesamte Befragung?

Minuten: _____

Wie Effizient konnte der Nutzer die Aufgaben ausführen (1 = sehr effizient, keine Fehleingaben/ -klicks, 2 = nomal, 3 = mehrere Fehleingaben)?

		1	**2**	**3**
1.	Aufgabe	☐	☐	☐
2.	Aufgabe	☐	☐	☐
3.	Aufgabe	☐	☐	☐
4.	Aufgabe	☐	☐	☐
5.	Aufgabe	☐	☐	☐

Wie viele Fehler wurden während der Ausführung gemacht? Waren sie fatal (wurde Hilfe benötigt)? Wie oft musste in der Aufgabe neu begonnen werden?

		Fehleingaben	**fatal?**	**Neustarts**
1.	Aufgabe	_____	☐	_____
2.	Aufgabe	_____	☐	_____
3.	Aufgabe	_____	☐	_____
4.	Aufgabe	_____	☐	_____
5.	Aufgabe	_____	☐	_____

Sonstige Beobachtungen:

_____

_____

A.5 JAVADOC.XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project default="javadoc">
    <target name="javadoc">
        <javadoc access="public" author="true"
          classpath='libs/XStream.jar;libs/jaxen-1.1.3.jar;libs/
              sgfilter_v1_2.jar;libs/microlog4android-1.0.0.jar;libs/
              ImpOpenGis.jar;libs/dom4j.jar;libs/AndroidAWT.jar;C:\
              PATH_TO_SDK\android-sdk\platforms\android-8\android.jar;
              C:\PATH_TO_SDK\android-sdk\add-ons\
              addon_google_apis_google_inc_8\libs\maps.jar'
          destdir="doc" nodeprecated="false" nodeprecatedlist="false"
              noindex="false"
          nonavbar="false" notree="false"
          packagenames="de.tud.android.mapbiq.locator.processing.
              fingerprint,org.placelab.mapper,org.placelab.client.
              tracker,org.placelab.util,org.placelab.core,org.placelab.
              midp,org.placelab.collections,de.tud.android.mapbiq.
              router,de.javagis.jgis.geometry,de.tud.android.mapbiq.
              core.models,de.tud.android.mapbiq.locator,de.tud.android.
              mapbiq.loader,de.tud.android.mapbiq.loader.processing,org
              .placelab.eventsystem,org.placelab.spotter,de.tud.android
              .mapbiq.utils,de.tud.android.mapbiq.locator.processing.
              combi,de.tud.android.mapbiq.locator.test,de.tud.android.
              mapbiq.utils.Jama.util,de.tud.android.mapbiq.core,de.tud.
              android.mapbiq.core.wfs,de.tud.android.mapbiq.locator.
              positioning,de.tud.android.mapbiq.locator.input,de.tud.
              android.mapbiq.renderer,de.tud.android.mapbiq.utils.Jama,
              de.javagis.jgis.util,de.tud.android.mapbiq.locator.
              processing.inertial,de.tud.android.mapbiq.locator.
              processing.wlan"
          source="1.6" sourcepath="gen;src" splitindex="true" use="true
              " version="true">
            <link href="file:/C:/PATH_TO_SDK/android-sdk/docs/reference
                "/>
            <link href="file:/C:/PATH_TO_PROJECT/MapBiquitous/doc/"/>
            <link href="file:/C:/PATH_TO_SDK/android-sdk/add-ons/
                addon_google_apis_google_inc_8/docs/reference"/>
        </javadoc>
    </target>
</project>
```

ABBILDUNGSVERZEICHNIS

2.1 Verbreitung von Mobiltelefonen und mobilen Anwendungen ([JES11]) 17

2.2 UML Klassendiagramm des Filter-Schemas 26

2.3 Architektur des MapBiquitous-Projekts (aus [Spr11]) 28

2.4 Layermodell der Fakultät Informatik TU Dresden (aus [Spr11]) 29

2.5 Ortsmodell des MapBiquitous-Projekts (aus [Sch09a]) 30

3.1 Google Maps Desktopversion, Kartenausschnitt 36

3.2 zwei Kartenausschnitte mit verschiedenem Zoomlevel auf Indoor-Kartierungen . . 36

3.3 drei Screenshots der Google Maps Android App 37

3.4 zwei Screenshots der Bing Maps Android App 43

3.5 drei Screenshots der Yahoo! Android App 45

3.6 drei Screenshots der ape@map Android App 47

5.1 allgemeine Abläufe in MapBiquitous 57

5.2 Client side Modells (Ausschnitt) 59

5.3 Darstellung des MVC-Pattern als Teil der Architektur 60

5.4 Package-Architektur 61

5.5 Anwendungsfall 63

5.6 Feature Request an den Server und Response 64
5.7 Feature editieren und an den Server senden 65
5.8 mögliche Nutzerinteraktionen in MapBiquitous 67
5.9 Touchscreen-Pattern aus [VWW10] 67
5.10 Touchscreen-Pattern aus [VWW10] 68
5.11 Mockups zur Kartenansicht 69
5.12 Mockups zu Menü, Einstellungen und Suche 70
5.13 Mockups zur Karten-, Listen- und Detailansicht 71
5.14 Mockups zur Kartenansicht 72
5.15 Diagramm - Auswertung Frage 2 74
5.16 Diagramme der Farbschemaevaluation 74

6.1 die wichtigsten Packages mit ausgewählten Klassen 81
6.2 loader-Package mit den asynchronen Requests (Ausschnitt) 85
6.3 drei Screenshot-Ausschnitte des Androidprototypen 88
6.4 Klassendiagramm des InfoBallons 89
6.5 zwei Screenshot-Ausschnitte des Androidprototypen 90

7.1 gefundene Usability-Probleme in Bezug auf Anzahl der Testpersonen (siehe [Nie00]) 99

TABELLENVERZEICHNIS

2.1 Klassifizierung der betrachteten Dienste nach [Bla07], [GNSW06], [GFDT05] 19

3.1 Vergleich der populärsten LBS-Communities 49

5.1 Farbdefinition für zeichenbare Objekte 73

7.1 Ergebnisse der SUS-Fragen 101

7.2 Ergebnisse der allgemeinen Fragen 102

CODEBEISPIELE

2.1 OWS Request 24

2.2 Ausschnitt der aktivierten Filter aus dem GetCapabilities Response 26

2.3 Request mit DWithin Filter des WFS 27

2.4 DWithin Filter des WFS in XML-Syntax 27

3.1 Google Data Protocol: Request nach der Ressource „myFeed“ 38

3.2 Google Data Protocol: Response für den Request 3.1 38

3.3 Google Data Protocol: Einfügen eines Eintrags 39

3.4 Google Data Protocol: Freitextsuche in der Ressource „myFeed“ 39

3.5 Google Data Protocol: Ergebnis der Suche 3.4 mit einem Ergebnis 40

3.6 Google Places: Ortssuchanfrage 41

3.7 Google Places: Antwort auf 3.6 41

3.8 Google Places: Ortsdetailanfragen 41

3.9 Google Places: Antwort auf 3.8 41

3.10 Google Geocoding: Adressanfrage 42

3.11 Microsoft Bing: Location Ressource 44

3.12 Yahoo! Placefinder: Ergebnisliste mit einem Eintrag 45

6.1 Ausschnitt aus der config.xml 83

6.2 GetFeature-Request mit POST-Daten 86

6.3 Initialisierung und Konfiguration des InfoBallonOverlay 88

appendix/javadoc.xml 121

LITERATURVERZEICHNIS

[AM11] Anas Aloudat and Katina Michael. Toward the regulation of ubiquitous mobile government: a case study on location-based emergency services in australia. *Electronic Commerce Research*, 11:31–74, 2011. 10.1007/s10660-010-9070-0.

[And09] Christine Andrae. *Spatial Schema : ISO 19107 und ISO 19137 vorgestellt und erklärt*. Wichmann, Heidelberg, 2009.

[And11] Christine Andrae. *Web portrayal services : OpenGIS Web Map Service, Styled Layer Descriptor, Symbology Encoding und ISO 19117 portrayal vorgestellt und erläutert*. Wichmann, Berlin; Offenbach, 2011.

[Ber05] Lars [Hrsg.] Bernard. *Geodateninfrastruktur : Grundlagen und Anwendungen*. Wichmann, Heidelberg, 2005.

[BKM08] Aaron Bangor, Philip T. Kortum, and James T. Miller. An empirical evaluation of the system usability scale. *International Journal of Human-Computer Interaction*, 2008.

[Bla07] Jörg Blankenbach. *Handbuch der mobilen Geoinformation : Architektur und Umsetzung mobiler standortbezogener Anwendungen und Dienste unter Berücksichtigung von Interoperabilität*. Wichmann, Heidelberg, 2007.

[BN06] Jürgen Bortz and Döring Nicola. *Forschungsmethoden und Evaluation für Human- und Sozialwissenschaftler*. Springer, 2006.

[Bri08] Thomas Brinkhoff. *Geodatenbanksysteme in Theorie und Praxis : Einführung in objektrelationale Geodatenbanken unter besonderer Berücksichtigung von Oracle Spatial*. Wichmann, Heidelberg, 2., überarb. und erw. aufl. edition, 2008.

[Bro96] J. Brook. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 1996.

[Bur97] Ph.D. Burbeck, Steve. Applications programming in smalltalk-80(tm): How to use model-view-controller (mvc). *The UIUC Smalltalk Archive*, March 1997.

[Com08] Portable Application Standards Committee. Portable operating system interface, 2008.

[DHdLD10] Anind Dey, Jeffrey Hightower, Eyal de Lara, and Nigel Davies. Location-based services. 9(1), 2010.

[DVLK11] Robert Deng, Jari Veijalainen, Shiguo Lian, and Dimitris Kanellopoulos. Editorial: special issue on ubiquitous electronic commerce systems. *Electronic Commerce Research*, 11:1–4, 2011. 10.1007/s10660-010-9071-z.

[EJ10] Roman Egger and Mario Jooss. *mTourism : mobile Dienste im Tourismus*. Gabler, Wiesbaden, 1. aufl. edition, 2010.

[Fou05] Open Software Foundation. Information technology open systems interconnection remote procedure call (rpc), 2005.

[fS01] International Organization for Standardization. Software engineering - product quality - part 1: Quality model, 2001.

[GFDT05] Kostas Gratsias, Elias Frentzos, Vasilis Delis, and Yannis Theodoridis. Towards a taxonomy of location based services. In Ki-Joune Li and Christelle Vangenot, editors, *Web and Wireless Geographical Information Systems*, volume 3833 of *Lecture Notes in Computer Science*, pages 19–30. Springer Berlin / Heidelberg, 2005.

[GHVJ09] Erich Gamma, Richard Helm, John Vlissides, and Ralph Johnson. *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley Publishing Company, 2009.

[GNSW06] Michael Grossniklaus, Moira Norrie, Beat Signer, and Nadir Weibel. Putting location-based services on the map. In James Carswell and Taro Tezuka, editors, *Web and Wireless Geographical Information Systems*, volume 4295 of *Lecture Notes in Computer Science*, pages 1–11. Springer Berlin / Heidelberg, 2006.

[HPSG11] Jana Hesselbarth, Katharina Papendick, Matthias C. Schroeder, and Annekathrin Gröninger. Usability quartett, 2011.

[Int03] International Organization for Standardization. Geographic Information - Spatial Schema, 2003.

[Int05] International Organization for Standardization. Geographic Information - Rules for application schema, 2005.

[JES11] JESS3. The geosocial universe 2.0. *jess3.com*, May 2011.

[JW08] Iris A. Junglas and Richard T. Watson. Location-based services. 51(3), March 2008.

[Keß11] Christoph Keßler. Konzeption und umsetzung einer gebäudeübergreifenden indoor/outdoor-navigation für das mapbiquitous system. Master's thesis, Technische Universität Dresden, Dresden, May 2011.

[Kom08] Klaus Komenda. *Location Based Services : am Beispiel eines mobilen Lokalführers*. VDM Verl. Dr. Müller, Saarbrücken, 2008.

[Küp05] Axel Küpper. *Location based services : fundamentals and operation*. John Wiley & Sons, Ltd., Chichester, 2005.

[Lar02] Craig Larman. *Applying UML and patterns: an introduction to object-oriented analysis and design and the unified process*. Prentice Hall Professional, 2002.

[Lik32] R. A. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 1932.

[LMK08] Miguel A. Labrador, Katina Michael, and Axel Küpper. Advanced location-based services. *Computer Communications*, 31(6):1053 – 1054, 2008. Advanced Location-Based Services.

[Nie00] Jakob Nielsen. Why you only need to test with 5 users. *http://www.useit.com/alertbox/20000319.html*, 2000.

[Ope05] Open Geospatial Consortium. OpenGIS Filter Encoding Implementation Specification, 2005.

[PW11] Krassie Petrova and Bin Wang. Location-based services deployment and demand: a roadmap model. *Electronic Commerce Research*, 11:5–29, 2011. 10.1007/s10660-010-9068-7.

[Sau11] Jeff Sauro. Measuring usability with the system usability scale (sus). *http://www.measuringusability.com/sus.php*, 2011.

[Sch09a] Jan Scholze. Entwicklung eines integrierten ansatzes zur indoor- und outdoor-positionsbestimmung auf basis einer föderation von gebäudedatenservern. Master's thesis, Technische Universität Dresden, Dresden, Sep 2009.

[Sch09b] Jan Scholze. Entwicklung eines konzeptes zur integration von gebäudeinformationen in/mit gis-systemen. Master's thesis, Technische Universität Dresden, Dresden, Feb 2009.

[Spr11] Thomas Springer. Mapbiquitous - an approach for integrated indoor/outdoor location-based services. 2011.

[SS10] Jan Scholze and Thomas Springer. Dokumentation und benutzerhandbuch zum projekt campus range location system, 2010.

[Tod11] Tim R. Todish. Not your parent's mobile phone: Ux design guidelines for smartphones. *Smashing Magazine*, 2011.

[TS04] Carsten Thelen and Jan Philipp Seng. Entwurfsmuster ... in java. Master's thesis, Aachen, 2004.

[VWW10] Craig Villamor, Dan Willis, and Luke Wroblewski. Touch gesture reference guide. *LUKEW IDEATION + DESIGN*, 2010. `http://www.lukew.com/ff/entry.asp?1071`.

DANKSAGUNG

Ich möchte mich an dieser Stelle bei allen bedanken, die mich in den letzten sechs Monaten bei meiner Diplomarbeit unterstützt haben. Ganz besonderer Dank gilt dabei meinem Betreuer, Thomas Springer, da er immer hilfreiches Feedback geben konnte. Die regelmäßigen Treffen und Diskussionen haben mir sehr geholfen.

Mein Dank gilt dabei auch Sina Grunau und Christoph Keßler, die die Entwicklung des Frameworks begleitet haben und durch ihre eigenen Arbeiten einen großen Teil zum MapBiquitous-Projekt beitragen. Der aktive Austausch und die gegenseitige Hilfestellung hat zur Lösung mancher Probleme beigetragen.

Ich danke auch allen Lehrstuhlmitarbeitern, Kommilitonen, Freunden und Freiwilligen, die bei der Evaluation mitgemacht und wertvolle Kritiken geäußert haben.

An diesem Punkt möchte ich auch die jahrelange Unterstützung meiner Familie während meines Studiums erwähnen und für die Geduld und das vermittelte Vertrauen Danke sagen.

Abschließend danke ich Karina und Georg, die sich trotz der Geburt Jariks, um die vielen fehlenden Kommas und einen besseren Ausdruck in meiner Arbeit bemüht haben.

ERKLÄRUNG

Ich erkläre, dass ich die vorliegende Arbeit selbständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Dresden, 14. März 2012